(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 043 664 A3

(12) **EUROPEAN PATENT APPLICATION**

(88) Date of publication A3:
15.06.2005 Bulletin 2005/24

(51) Int Cl.7: **G06F 17/30**

(43) Date of publication A2:
11.10.2000 Bulletin 2000/41

(21) Application number: **00105229.9**

(22) Date of filing: **13.03.2000**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **06.04.1999 US 287390**

(71) Applicant: **SHARP KABUSHIKI KAISHA**
Osaka 545-8522 (JP)

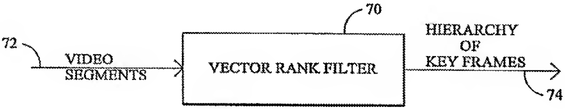
(72) Inventor: **Crinon, Regis J.**
Camas, WA 98607 (US)

(74) Representative: **Müller - Hoffmann & Partner Patentanwälte,**
Innere Wiener Strasse 17
81667 München (DE)

(54) **Video skimming system utilizing the vector rank filter**

(57) Automated summarization of digital video sequences is accomplished using a vector rank filter (70). The consecutive frames of a digital video sequence can be represented as feature vectors which are successively accumulated in a set of vectors. The distortion of the set by the addition of each successive vector or the cumulative distance from each successive vector to all other vectors in the set is determined by a vector rank filter (70). When the distortion exceeds a threshold value the end of a video segment is detected. Each frame in a video segment can be ranked according to its relative similarity to the other frames of the set by applying the vector rank filter (70) to the feature vectors representing the video frames. To produce a summary of a video sequence which is most representative of the content of the sequence, frames are chosen that correspond to vectors that are the least distant to or produce the least distortion of the set of vectors representing the segment. The ranking of the relative distortion can be used as the basis for selecting more than one frame from each segment to produce a hierarchy of summaries containing greater numbers of the frames having the most representative content.

FIG.5



EP 1 043 664 A3

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 00 10 5229

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
X	JU S X ET AL: "Analysis of gesture and action in technical talks for video indexing" COMPUTER VISION AND PATTERN RECOGNITION, 1997. PROCEEDINGS., 1997 IEEE COMPUTER SOCIETY CONFERENCE ON SAN JUAN, PUERTO RICO 17-19 JUNE 1997, LOS ALAMITOS, CA, USA,IEEE COMPUT. SOC, US, 17 June 1997 (1997-06-17), pages 595-601, XP010237585 ISBN: 0-8186-7822-4 * page 595 - page 599 *	1-60	G06F17/30
X	GUNSEL B ET AL: "Content-based video abstraction" IMAGE PROCESSING, 1998. ICIP 98. PROCEEDINGS. 1998 INTERNATIONAL CONFERENCE ON CHICAGO, IL, USA 4-7 OCT. 1998, LOS ALAMITOS, CA, USA,IEEE COMPUT. SOC, US, vol. 3, 4 October 1998 (1998-10-04), pages 128-132, XP010586749 ISBN: 0-8186-8821-1 * the whole document *	1-60	TECHNICAL FIELDS SEARCHED (Int.Cl.7) G06F
X	US 5 864 366 A (YEO ET AL) 26 January 1999 (1999-01-26) * abstract * * column 4, line 58 - column 9, line 11 *	1-60	
D,X	US 5 635 982 A (ZHANG ET AL) 3 June 1997 (1997-06-03) * abstract * * column 1, line 56 - column 7, line 62 *	1-60	

-/--

The present search report has been drawn up for all claims

1

Place of search	Date of completion of the search	Examiner
The Hague	11 April 2005	Dumitrescu, C

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 00 10 5229

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
A	LUCAT L ET AL: "Vector-median type filters and fast-computation algorithms" CIRCUITS AND SYSTEMS, 1997. ISCAS '97., PROCEEDINGS OF 1997 IEEE INTERNATIONAL SYMPOSIUM ON HONG KONG 9-12 JUNE 1997, NEW YORK, NY, USA,IEEE, US, vol. 4, 9 June 1997 (1997-06-09), pages 2469-2472, XP010236228 ISBN: 0-7803-3583-X * the whole document * -----	1-60	
			TECHNICAL FIELDS SEARCHED (Int.Cl.7)

The present search report has been drawn up for all claims

1

Place of search	Date of completion of the search	Examiner
The Hague	11 April 2005	Dumitrescu, C

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 00 10 5229

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report.
The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

11-04-2005

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5864366 A	26-01-1999	JP 3074648 B2	07-08-2000
		JP 10294931 A	04-11-1998
US 5635982 A	03-06-1997	EP 0690413 A2	03-01-1996
		JP 8237549 A	13-09-1996

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 043 664 A2

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
11.10.2000 Bulletin 2000/41

(51) Int. Cl.[7]: **G06F 17/30**

(21) Application number: **00105229.9**

(22) Date of filing: **13.03.2000**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **06.04.1999 US 287390**

(71) Applicant:
SHARP KABUSHIKI KAISHA
Osaka 545-8522 (JP)

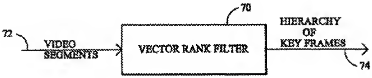
(72) Inventor: **Crinon, Regis J.**
Camas, WA 98607 (US)

(74) Representative:
MÜLLER & HOFFMANN Patentanwälte
Innere Wiener Strasse 17
81667 München (DE)

(54) Video skimming system utilizing the vector rank filter

(57) Automated summarization of digital video sequences is accomplished using a vector rank filter (70). The consecutive frames of a digital video sequence can be represented as feature vectors which are successively accumulated in a set of vectors. The distortion of the set by the addition of each successive vector or the cumulative distance from each successive vector to all other vectors in the set is determined by a vector rank filter (70). When the distortion exceeds a threshold value the end of a video segment is detected. Each frame in a video segment can be ranked according to its relative similarity to the other frames of the set by applying the vector rank filter (70) to the feature vectors representing the video frames. To produce a summary of a video sequence which is most representative of the content of the sequence, frames are chosen that correspond to vectors that are the least distant to or produce the least distortion of the set of vectors representing the segment. The ranking of the relative distortion can be used as the basis for selecting more than one frame from each segment to produce a hierarchy of summaries containing greater numbers of the frames having the most representative content.

FIG.5



EP 1 043 664 A2

Description

BACKGROUND OF THE INVENTION

[0001] The present invention relates to digital video content analysis and more particularly to a system for summarizing digital video sequences as a series of representative key frames.

[0002] The increasing availability and use of video have created a need for video summaries and abstractions to aid users in effective and efficient browsing of potentially thousands of hours of video. Automation of video content analysis and extraction of key representative content to create summaries has increased in significance as video has evolved from an analog to a digital format. Digital television, digital video libraries, and the Internet are applications where an appliance that can "view" the video and automatically summarize its content might be useful.

[0003] Generally, a sequence of video includes a series of scenes. Each scene, in turn, includes a series of adjoining video "shots." A shot is a relatively homogeneous series of individual frames produced by a single camera focusing on an object or objects of interest belonging to the same scene. Generally, automated video content analysis and extraction involve "viewing" The video sequence, dividing The sequence into a series of shots, and selecting one or more "key frames" from each of the shots to represent the content of the shot. A summary of the video sequence results when the series of key frames is displayed. The summary of the video will best represent the video sequence if the frames which are most representative of the content of each shot are selected as key frames for inclusion in the summary. Creation of a hierarchy of summaries, including a greater or lesser number of key frames from each shot, is also desirable to satisfy the differing needs of users of the video.

[0004] The first step in the summarization process has been the division of the video into a series of shots of relatively homogeneous content. Video shot transitions can be characterized by anything from abrupt transitions occurring between two consecutive frames (cuts) to more gradual transitions, such as "fades," "dissolves," and "wipes."' One technique for detecting the boundaries of a shot involves counting either the number of pixels or the number of predefined areas of an image that change in value by more than a predefined threshold in a subsequent frame. When either the total number of pixels or areas satisfying this first criterion exceeds a second predefined threshold a shot boundary is declared. Statistical measures of the values of pixels in pre-specified areas of the frame have also been utilized for shot boundary detection. Pixel difference techniques can be sensitive to camera and object motion. Statistical techniques tend to be relatively slow due to the complexity of computing the statistical formulas.

[0005] Histograms and histogram related statistics are the most common image representations used in shot boundary detection. Gray level histograms, color histograms, or histogram related statistics can be compared for successive frames. If the difference exceeds a predefined threshold, a shot boundary is detected. A second threshold test may also be included to detect the more gradual forms of shot transition.

[0006] Selecting one or more key frames which best represent the relatively homogeneous frames of a shot has been more problematic than defining shot boundaries. Lagendijk et al. in a paper entitled VISUAL SEARCH IN A SMASH SYSTEM, Proceedings of the International Conference on Image Processing, pages 671-674, 1996, describe a process in which shot boundaries are determined by monitoring cumulative image histogram differences over time. The frames of each shot are temporally divided into groups reflecting the pre-specified number of key frames to be extracted from each shot. The frame at the middle of each group of frames is then selected as the key frame for that group. The selection of a key frame is arbitrary and may not represent the most "important' or "meaningful" frame of the group. Also, this process must be performed "off-line" with storage of the entire video for "review" and establishment of shot boundaries, followed by temporal segmentation of shots and then extraction of key frames. For key frame extraction, the stored video must be loaded into a processing buffer so that the group of frames and associated key frames can be calculated. The size of a shot is limited by the size of the processing buffer.

[0007] In the copending application of Ratakonda, Serial No. 08/994,558, filed December 19, 1997, shot boundaries are determined by monitoring variations in the differences in image histograms over time. Individual shots are further partitioned into segments which represent highly homogeneous groups of frames. The partitioning of shots into segments is achieved through an iterative optimization process. For each video segment, the frame differing most from the key frame of the prior segment is selected as the next key frame of the summary. A key frame is selected on The basis of the frame's difference from the prior key frame and not on the basis of its representation of the other frames belonging to the segment. Like the technique proposed by Lagendijk, this technique must be performed off-line and an entire video shot must be stored for review, segment partitioning, and key frame selection. Additional memory is required to store the prior key frame for comparison.

[0008] Zhang et al., U.S. Patent No. 5,635,982, disclose a method in which the difference between frames is monitored and accumulated. When the accumulated difference exceeds a predefined threshold, a potential key frame is detected. The potential key frame is designated as a key frame if, in addition, the difference between the potential key frame and the previous key frame exceeds a preset threshold. Without additional processing, the locations of key frames always coincide with the beginning a new shot.

[0009] Smith et al., in a paper entitled VIDEO SKIMMING AND CHARACTERIZATION THROUGH THE COMBINATION OF IMAGE AND LANGUAGE UNDERSTANDING TECHNIQUES and Mauldin et al., U.S. Patent No. 5,664,227 disclose an elaborate key frame identification technique based on context rules related to repetitiveness, degrees of motion, and audio and video content. The key frame sequences can be used to provide compact summaries of video sequences but the method is complex and does not support creation of hierarchical video summaries.

[0010] What is desired is a technique of automated video content analysis and key frame extraction which selects key frames that are the most representative frames of each shot or segment of the video sequence. Simple implementation, conservation of computational resources, and the ability to accept a variety of inputs are desirable characteristics of such a technique. It is desired that the technique provide for content analysis and key frame extraction both "on-line (in real time)," without the need to store the entire video sequence, and "off-line." Further, a technique of conveniently creating a hierarchy of summaries, each successively containing a smaller number of the most representative frames, is desired.

SUMMARY OF THE INVENTION

[0011] The present invention overcomes the aforementioned drawbacks of the prior art by providing a method and apparatus for digital video content analysis and extraction based on analysis of feature vectors corresponding to the frames of a video sequence. In the first embodiment of the invention, a method is provided for identifying a key video frame within a segment of video having frames of relatively homogeneous content including the steps of characterizing each video frame as a feature vector identifying a key feature vector that minimizes the distortion of the group of feature vectors; and identifying the video frame corresponding to the key feature vector as the key video frame. Key frames selected by the method of this first embodiment of the present invention are the frames which are the most representative of the content of the set of frames in each shot of a sequence.

[0012] In the second embodiment a method is provided for determining the boundaries of a video segment within a video sequence comprising the steps of defining a threshold distortion; locating a first frame in the video segment; defining a first feature vector representative of the first frame; including the first feature vector in a set of segment feature vectors; defining a next feature vector representative of a subsequent video frame; including the next feature vector in the set of segment feature vectors; calculating the distortion of the set of segment feature vectors resulting from including the next feature vector in the set; and comparing the distortion of the set of segment feature vectors with the threshold distortion. The steps of characterizing subsequent frames as feature vectors, adding feature vectors to the set, calculating the distortion, and comparing the distortion with the threshold is repeated until the distortion of the set of segment feature vectors has achieved some predefined relationship to the threshold distortion thereby defining the second boundary of the segment. Prior receipt and storage of the entire video sequence are not required for the segmentation process. Key frames can be identified simultaneously with segmentation of the video by applying the methods of both the first or third embodiments.

[0013] In the third embodiment of the present invention a method is provided for creating summaries of video sequences including more than one key frame from each segment comprising the steps of dividing the video frames of the sequence into at least one video segment of relatively homogeneous content including at least one video frame; defining a feature vector representative of each of the video frames; ranking the feature vectors representing the frames included in each video segment according to the relative distortion produced in the set of feature vectors representing the segment by each feature vector included in the set; and including in the summary of the sequence, video frames represented by the feature vectors producing relative distortion of specified ranks. Utilizing the method of this third embodiment, a hierarchy of key frames can be identified from which hierarchical summaries of a video sequence can be created with each summary including a greater number of the most representative frames from each segment.

[0014] The foregoing and other objectives, features and advantages of the invention will be more readily understood upon consideration of the following detailed description of the invention, taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015]

FIG. 1 illustrates the method of identifying key frames of the first embodiment of the present invention.

FIG. 2A is a flow chart illustrating one technique for performing the method of key frame identification of the first embodiment of the invention.

FIG. 2B is a continuation of the flow chart of FIG. 2A.

FIG. 2C is a continuation of the flow chart of FIG. 2B.

FIG. 2D is a continuation of the flow chart of FIG. 2C.

FIG. 3 illustrates the method of identifying segment boundaries and key frames of the second embodiment of the present invention.
FIG. 4A is a flow chart illustrating one technique for performing the method of segment boundary and key frame identification of the second embodiment of the invention.
FIG. 4B is a continuation of the flow chart of FIG. 4A.
FIG. 4C is a continuation of the flow chart of FIG. 4B.
FIG. 4D is a continuation of the flow chart of FIG. 4C.
FIG. 5 illustrates the method of identifying a hierarchy of key frames of the third embodiment of the present invention.
FIG. 6 is a schematic representation of an exemplary video sequence where each frame within a segment has been ranked according to the relative distortion of the segment produced by the frame.
FIG. 7A is a flow chart illustrating one technique for performing the method of key frame ranking and compilation of hierarchical summaries of the third embodiment of the invention.
FIG. 7B is a continuation of the flow chart of FIG. 7A.
FIG. 7C is a continuation of the flow chart of FIG. 7B.
FIG. 7D is a continuation of the flow chart of FIG. 7C.
FIG. 7E is a continuation of the flow chart of FIG. 7D.
FIG. 8 is a schematic illustration of video sequence summarizing appliance.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0016] Generally, a video sequence includes a series of scenes. Each scene, in turn, includes a series of adjoining video segments or "shots." A shot is a relatively homogeneous series of consecutive individual frames produced by a single camera focusing on an object or objects of interest Generally, automated video content analysis and extraction involve "viewing" the video sequence, dividing the sequence into a series of shots, and selecting of one or more "key frames" to represent the content of each of the shots. Displaying or providing a list of the key frames summarizes the video sequence.

[0017] The content of a frame of digital video can be represented in several ways. A common representation is an image histogram obtained by calculating the frequency distribution of picture elements or picture areas as a function of some property of the element or area. For example, a color histogram of an image is the frequency distribution of picture elements or areas as a function of the color of the element or area. The image histogram can, in turn, be represented as a vector signal including a combination of separate components each carrying information about a different property of the histogram, otherwise known as a "feature vector." Other image representations can also be captured by feature vectors. Feature vectors, for example, can be used to represent the average intensity of an image; an ordered set of image samples; an ordered, fixed set or subset of linear or nonlinear combinations of samples from an image; a luminance, chrominance, or combined luminance and chrominance sample value histogram; or an ordered set of statistical or deterministic, or a combination of statistical and deterministic, parameter values from an image or an image histogram.

[0018] In the present invention, the frames of a video sequence are preferably represented as multidimensional feature vectors. A vector rank filter or a more specialized version of the vector rank filter, the vector median filter, is used to determine the relative cumulative distances from each feature vector in a set to the other feature vectors in the set. The cumulative distance from a feature vector to all other feature vectors in the set measures the "distortion" of the set of vectors as evaluated from the vector under investigation. The distortion indicates the homogeneity of the content of a frame of video, characterized by the corresponding feature vector, with the content of all other frames of video in the set The output of the vector rank filter is used to resolve the homogeneity of the video's content. The vector rank filter permits identification of the video frame that is most representative of the content of the frames of a set. It also facilitates determining when the content of a next frame in a series differs substantially from the preceding frames in the series or ranking the frames of a set according to the relative homogeneity of content of the frames or distortion of the set by each frame. Finally, the vector rank filter enables an on-line implementation of a video summarizer, meaning that representative video frames are generated as more video frames or video fields are input into the system. An on-line implementation offers advantages over an off-line implementation where typically all video frames must be acquired and stored before they can be processed. The biggest advantage may be in the fact that a "smart" receiver capable of monitoring and evaluating its own resources (computing resources, memory resources, communication bandwidth resources,...) may choose to tune the parameters of the on-line summarizing algorithm in response to instantaneous variations of the resources. In effect, an on-line implementation allows a receiver to make smart trade-offs in time thereby producing video summarization results which may be viewed as optimal results with respect to video content and resource availability at the time of its processing.

[0019] The vector rank filter can be used to rank the vectors of a set of vectors in terms of the cumulative distance

from a vector to the other vectors of the set or the relative distortion of the set caused by each vector. Given a set of P vectors $\underline{x}_h$, where $1 \le h \le P$, the output of the vector rank filter will be a vector, $\underline{x}_{(h)}$, which has the hth smallest distortion value among the P distortion values. If $D_{(m)}{}^q$, where $1 \le m \le P$ and where q identifies the distance measure, is the distortion of rank m (ranked in increasing values of m) then $D_{(1)}{}^q$ corresponds to the minimum distortion, that is, distortion of rank one. Likewise, $D_{(2)}{}^q$ corresponds to the second smallest distortion. The distortion $D_{(P)}{}^q$ corresponds to the maximum distortion, and $D_{(h)}{}^q$ corresponds to the hth ranked smallest distortion. The vector characterized by the least cumulative distance is $\underline{x}_1$, the vector characterized by the second smallest cumulative distance is $\underline{x}_2$, and so forth. The vector characterized the greatest cumulative distance is $\underline{x}_P$. For the vector $\underline{x}_h$ the distortion equals:

$$D_{(h)}{}^q = \sum_{j=1}^{j=P} w_{j,(h)} \|\underline{x}_j - \underline{x}_{(h)}\|^q$$

where w is a weighting factor associated with the joint use of the vector $\underline{x}_j$ and the vector $\underline{x}_{(h)}$ and q specifies the distance or distortion measure. The choice of q may be determined by application considerations. For example, summing the absolute differences between the vectors (q=1) is computationally less intensive than summing the squares of the differences (q=2). In other applications, q might be selected to optimize the distortion with respect to certain input vector statistics. Likewise the weight $w_{j,(h)}$ might be selected to represent the relative importance of input feature vectors or to implement particular rank-ordered statistics.

[0020] In the case of ties in rank, the same rank may be allocated to all vectors sharing the same distortion value. In this case, the next rank used would be adjusted to reflect the fact that more than one vector occupies the "tied" rank. For example, if the distortions D_i^q, D_l^q, $D_m{}^q$ are equal and represent the second ranked distortion, the respective vectors $\underline{x}_i$, $\underline{x}_l$, and $\underline{x}_m$ can all be assigned the rank of three (the average of the rank values: two, three and four). The vector producing the next smallest distortion would be assigned the rank five to reflect the fact that three vectors are sharing ranks two, three and four. An alternative approach to dealing with ties would be to assign a fractional rank to vectors sharing the same distortion. For example, if two vectors produce equal distortion, ranked two, the vectors might be assigned the rank 5/2 [(2+3)/2 = 5/2] to indicate the tie.

[0021] A specialized vector rank filter, the vector median filter, is particularly useful in identifying the most representative frame of a video segment or shot. The vector rank filter can also be used in determining the boundaries of a shot or segment. The output of the vector median filter identifies the vector producing the minimum distortion among all vectors in a set of vectors. In other words, the vector median filter identifies the vector of rank one as identified by the vector rank filter. Given a set of P vectors $\underline{x}_j$, where $1 \le j \le P$, the output of the vector median filter, $\underline{x}_k$, is such that the index k satisfies:

$$k = \underset{1 \le i \le P}{\arg\min} \left\{ D_i^q \right\}$$

where the distortion or cumulative distance D_i^q, for q = 1, 2, . . . , ∞ is defined as

$$D_i{}^q = \sum_{j=1}^{j=P} w_{j,i} \|\underline{x}_j - \underline{x}_i\|^q$$

and where $w_{j,i}$ denotes a weighting factor applied when vector $\underline{x}_j$ and vector $\underline{x}_i$ are used jointly and q specifies the distortion or distance measure.

[0022] For example, if the vectors $\underline{x}_i$ are R-dimensional vectors with components $x_{i,r}$, where $1 \le i \le R$, and if the weight value, $w_{i,j}$ and the distortion or distance measure q are equal to 1 the minimum distortion is:

$$D_i{}^1 = \sum_{j=1}^{j=P} \sum_{r=1}^{r=R} |x_{j,r} - x_{i,r}|$$

For $q = 2$, the distortion is:

$$D_i^{\,2} = \sum_{j=1}^{j=P} \sum_{r=1}^{r=R} (x_{j,r} - x_{i,r})^2$$

For $q = \infty$, the distortion is:

$$D_i^{\,\infty} = \sum_{j=1}^{j=P} \max \{|x_{j,1} - x_{i,1}|, |x_{j,2} - x_{i,2}|, \ldots\ldots, |x_{j,R} - x_{i,R}|\}$$

[0023] The output of the vector median filter is the vector which is globally the "closest" relative to the distortion measure q, to all The other vectors within the set P vectors. The output vector, x_k, can be considered to be the most representative vector of all the P input vectors in the set since it is associated with the least distortion within the vector set. The video frame corresponding to this feature vector can be considered to be the most representative of the set of video frames associated with the set of input feature vectors.

[0024] Referring to FIG. 1, in a first embodiment of the present invention video segments 10 are input to a vector median filter 12. The vector median filter 12 identifies the key frames 14 that are the most representative of the video segment 10 where the boundaries of the segment 10 has been established by the methods of the present invention or otherwise. The key frame 14 from each of a plurality of segments may be used as a summary of the video sequence comprising that plurality of segments. For example, assume a jth segment of video with two identified time boundaries, times t(j) and t(j+1), where j = 1, 2, . . . ,N. The jth segment contains M frames of video which are denoted here as F(t(j)), F(t(j)+1), ...F(t(j)+M-1). The video frames can be defined as a set of R-dimensional feature vectors denoted by $\underline{h}$(t(j)), $\underline{h}$(t(j)+1), . . . $\underline{h}$(t(j)+M-1) where the vector $\underline{h}$(t(j)+i) is associated with the frame F(t(j)+i) and where $0 \le i \le M-1$. The next video segment (the (j+1)th segment) starts at time t(j+1) such that t(j+1)=t(j) +M where j<N.

[0025] The application of the vector median filter 12 to the vectors $\underline{h}$(t(j)), $\underline{h}$(t(j)+1), . . . $\underline{h}$(t(j)+M-1) permits identification of the vector producing the least distortion among the set of vectors corresponding to the video segment 10. This vector will correspond to the frame, belonging to the set of frames F(t(j)), F(t(j)+1), . . . F(t(j)+M-1), which is most representative of the content of the frames in the segment 10. A key frame 14 can be determined in the same fashion for each segment in the video sequence. The resulting series of key frames 14 constitutes a summary of the video sequence where each key frame is the most representative frame of each segment. While the number of frames M may vary from segment to segment the overall number of key frames identified by this embodiment can be controlled if the input frame rate is known.

[0026] FIGS. 2A Through 2D illustrate a flow chart for one technique of performing the method of key frame identification of the first embodiment of the present invention where the feature vector utilized is based on an image histogram. The variables used in the flow chart include: j - the number of the first frame of the segment; b - the segment number; acc_dif - the distortion or cumulative distance between the current frame and the prior frames under consideration; min_dif - the ,current minimum cumulative distortion; n - the current frame; m - the number of frames being considered (from the beginning of the segment to the current frame (n)); vm(b) - the vector median for the bth segment; P(b) - the number of frames in the bth segment, L - the number of segments in the video sequence and HUGE is a large number that cannot be obtained by any cumulative distance.

[0027] In the method for identifying a key frame of this embodiment:

[0028] First, a system for performing the method sets j and b to 1 (step S1). Then, the system computes $\underline{h}$(j), histogram of image j, and sets cumulative distortion ace_dif(j) to 0 (step S2). Next, from bth video segment starting at jth video frame the system sets min_dif to HUGE (step S3). Setting n = j + 1 (step S4), the system acquires image n, calculates histogram $\underline{h}$(n) and sets cumulative distortion acc_dif(n) to 0 (step S5). Setting m = j (step S6), the system computes dif, the sum of the absolute bin (histogram entries) value differences between histogram $\underline{h}$(m) and histogram $\underline{h}$(n) (step S7).

[0029] The system updates cumulative distortion values acc_dif(n) + and acc_dif(m) + for histograms n and m to dif (step S8). If acc_dif(m) < min_dif (step S9), the system sets min_dif = acc_dif(m) and vm(b) = m (step S10). If m = n - 1 (step S11), when acc_dif(n) <min_dif (step S13), the system sets min_dif = acc_dif(n) and vm(b) = n (step S14). If not, setting m = m + 1 (step S12), the process returns to step S7. Next, if n = P(b) (segment length P(b) assumed known) (step S15), when q = L (L is number of segments) (step S17), the system starts displaying sequences of representative frames (step S19) and sets b = 1 (step S20) and p = 1 (step S21). In step S15 if n is not equal to P(b), the system sets n = n + 1 (step S16) and returns to step S5. Instep S17 if q is not equal to L, the system sets b = b + 1 and j = n + 1 (step S18) and returns to step S2.

[0030] If p = vm(b) (step S22), the system display video frame (step S23). Next, if P = P(b) (step S24) and b = L (step 826), the system stops the process of the method, If P is not equal to P(b), the system sets p = p + 1 (step S25) and returns to step S22. In step S26, if b is not equal to L, the system sets b = b + 1 (step S27) and returns to step S21.

[0031] Referring to FIG. 3, in the second embodiment of the present invention a video processor 40 receives a video sequence 42. The video processor 40 produces an automatic selection of key frames 44, similar to FIG. 1, such that the cumulative frame difference between the key frame and the other frames within the segment does exceed a given level of distortion. This ensures that the overall distortion within a segment remains bounded relative to the most representative video frame. This criterion also provides a technique to determine the video segment boundaries 46 "on-line" simultaneously with the key frames 44.

[0032] In the second embodiment of the present invention the boundaries 46 of the video segments do not need to be pre-specified. To determine the boundaries 46 of a video segment, the boundaries 46 are designated as a variable which is calculated "on-line" as the each successive frame of video is received and "viewed." While video comprises consecutive frames, a sampling technique can be used to select frames which are subsequent but not necessarily consecutive. When the distortion or cumulative distance resulting from the addition of a last frame to the set exceeds some predefined threshold, for the first time, the boundary ending the segment is detected. For example, a consecutive set of video frames, F(t(j)), F(t(j)+1), ..., F(t(j+1)-1), F(t(j+1)), corresponding to a time period t(j), t(j)+1 ... t(j+1)-1, t(J+1), is to be segmented. A first segment boundary is assumed to have been established by this method at time t(j). The segment starting at time t(j) is segment j. Since video comprises a set of consecutive images, the first image of the first segment is assumed to correspond to the first frame of the sequence. The first segment boundary of any subsequent segment can be determined by successively locating the second boundary each segment which follows the first segment. As each successive frame is received, a feature vector is computed and added to the feature vector set beginning with $\underline{h}$(t(j))corresponding to frame F(t(j)). The minimum distortion of the set of vectors is evaluated with the vector median filter. If the cumulative minimum distance, $D^q_{(1)}$, calculated by the vector median filter when applied to frames F(t(j)), ..., F(t(j+1)), exceeds some predefined threshold T for the first time when frame F(t(j+1)) is added to the set of vectors, the non-inclusive, ending segment boundary for the segment j is declared to be at time t(j+1). In other words, the time t(j+1) represents the beginning of the next video segment (segment j+1) if the minimum cumulative distance is less than the threshold T ($D^q_{(1)}$(t(j+1) - 1) < T) for the frame immediately preceding this frame (frame F(t(j+1)-1)), but the minimum cumulative distortion exceeds the threshold T ($D^q_{(1)}$(t(j+1)) - 1 ≥ T) when frame F(t(j+1)) is added to the set.

[0033] As an alternative criterion, a shot boundary may be declared when the cumulative distance associated with the most recent frame exceeds some function of the minimum distance in the set of vectors, such as the product of the minimum distance and some factor (k) which is greater than one. In other words, a new segment is detected when:

$$D^q_{t(j+1)-1} < k\,D^q_{(1)}(t+(j+1)-1) \text{ and } D^q_{t(j+1)} < k\,D^q_{(1)}(t+(j+1)) \text{ where } k>1.$$

[0034] The threshold T or the factor k may be selected as a function of the noise power in the image or as a function of a target compression ratio to be achieved by the video summary or some combination of noise power and target compression ratio.

[0035] When segment boundaries are detected, the key frame for the segment is the output of the vector median filter applied to the frames of the segment as described in the first embodiment As a result summaries can be produced "on-line" where the shot boundaries are detected and a key frame within each shot is simultaneously selected as the frames of video are "viewed" without the requirement to store the entire video sequence for review and processing. Further, a shot containing a single frame can be identified by the method.

[0036] Referring to FIGS. 4A Through 4D, a flow chart illustrates a technique of implementing the method of the second embodiment of the present invention where the feature vectors are image histograms and a shot boundary is declared when the maximum cumulative distance in a candidate video segment exceeds the minimum distortion calculated by the vector median filter multiplied by distance multiplier k. The variables are the same as those described above for FIG. 2. An additional variable, previous_vm(b), is used to denote the index of the previously selected vector median for segment b. This variable records the position of the key frame once a segment boundary has been detected. In this Java code, the parameter is time-varying and decreasing in time such that a segment boundary is always generated by a certain, pre-defined number of frames. The Java code also includes a shot boundary detection which detects homogeneous portions of the input video sequence. Here is a shot that consists of several segments, each segment being represented by a keyframe The Java code also includes an additional mechanism monitoring how many times the same vector median value is consecutively selected. This mechanism is put in place to ensure that keyframes result from selecting the same vector median for some duration in time. Table A provides a sample source code written in the Java programming language for implementing this technique of the second embodiment.

[0037] In the method of this embodiment;

[0038] First, a system for performing the method sets j and b to 1 (step S31). Then, the system computes $\underline{h}$(j), histogram of image j (step S32), and sets cumulative distortion acc_dif(j) and previous_vm(b) to zero (step S33). Next,

from bth video segment starting at jth video frame the system sets min_dif = HUGE (step S34). Selling n = j + 1 (step S35), if it is end of video sequence (step S36), the system stores P(b) and sets L = b (step S37). If not, the system acquires image n, calculates histogram h(n) and sets cumulative distortion acc_dif(n) to 0 (step S38). Selling m = j (step S39), the system computes dif, the sum of the absolute bin (histogram entries) value differences between histogram h(m) and histogram h(n) (step S40).

[0039] The system updates cumulative distortion values acc_dif(n) + and acc_dif(m) + for histograms n and m to dif (step S41). If acc_dif(m) < min_dif (step S42), the system sets min_dif = acc_dif(m) and vm(b) = m (step S43). If m = n - 1 (step S44), when acc_dif(n) < min_dif (step S46), the system sets min_dif = acc_dif(n) and vm(b) = n (step S47). In step S44, if m is not equal to n - 1, setting m = m + 1 (step S45), the process returns to step S40. Next, if acc_dif(n) > (k*min_dif) (step S48), the system sets end of segment to n-1, stores P(b), P(b) = n-j and vm(b) = previous vm(b) (step S50). Then setting b = b + 1 and j = n (step S51), the process returns to step S33. If not (step S48), setting n = n +1 and previous_vm(b) = vm(b) (step S49), the process returns to step S36.

[0040] Next, the system starts displaying sequences of representative frames (step S52). Setting b = 1 (step S53) and p = 1 (step S54), if p = vm(b) (step S55), the system displays video frame (step S56). Next, if P = P(b) (step S57) and b=L (step S59), the system stops process of the method, If P is not equal to P(b), the system sets p = p + 1 (step S58) and returns to step S55. In step S59, if b is not equal to L, the system sets b = b + 1 (step S60) and to step S54.

[0041] Referring to FIG. 5, in a third embodiment of the present invention a hierarchical process provides a video skimming methodology based on the frame ranking results produced by a vector rank filter 70. The process identifies a hierarchy of frames 74 within a video segment 72 which is the input to the vector rank filter 70. Hierarchical summaries can be constructed by skimming different numbers of key frames from each video segment on the basis of the key frame's rank in the hierarchy of key frames 74. The method can easily produce a hierarchical series of summaries with each successive summary containing a greater number of the most representative frames from each shot. The result is a video content driven temporal sub-sampling mechanism based on the rank values produced by the vector rank filter. FIG. 6 illustrates an exemplary video sequence of eleven frames 60 which has been segmented into four segments 62 by the methods of the present invention or otherwise. The vector rank filter is used to generate the rank 64 of the frames 60 within each segment 62 according to the distortion or cumulative distance produced by the frame 60 in the set of the frames of the segment 62. To skim the video, frames 60 can be selected for inclusion in a summary on the basis of the relative cumulative distance produced by the frame in the video segment 62 of which the frame 60 is a member. Skimming the video at level one, would include the frames of rank one 66 in the coarsest summary. A level two summary could include frames ranked one 66 and two 68. At each level of the hierarchy, additional frames 60 would be assigned to the summary until all of the frames 60 of the sequence are included in the summary of maximum rank. While each increase in rank adds frames 60 to the summary which are less representative of the content of the segment 62, each summary comprises those frames 60 which are the most representative of the sequence. FIGS. 7A through 7E illustrate a flow chart of a technique of implementing this third embodiment of the present invention. The variables used in FIGS. 7A through 7E are those identified above for FIG. 4.

[0042] In the method of this embodiment;

[0043] First, a system for performing the method sets j and b to 1, and pmax to 0 (step S71). Then the system computes h(i),histogram of image j (step S72) and sets cumulative distortion acc_dif(j) to zero and previous_vm(b) = j (step S73). Next, from bth video segment starting at jth video frame the system sets min_dif = HUGE (step S74). Selling n = j +1 (step S75), if it is end of video sequence (step S76), the system stores P(b) and sets L = b (step S77). If not (step S76), the system acquires image n, calculates histogram h(n) and sets cumulative distortion acc_dif(n)to 0 (step S78). Selling m = j (step S79), the system computes dif, the sum of the absolute bin (histogram entries) value differences between histogram h(m) and histogram h(n) (step S80).

[0044] The system updates cumulative distortion values acc_dif(n) + and acc_dif(m) + for histograms n and m to dif (step S81). If acc_dif(m) < min_dif (step S82), the system sets min_dif = acc_dif(m) and vm(b) = m (step S83). Next, if m = n - 1 (step S84), when acc_dif(n) < min_dif (step S86), the system sets min_dif = acc_dif(n) and vm(b) = n (step S87). If not (step S84), selling m = m + 1 (step S85), the process returns to step S80. Next, in step S88, if acc_dif(n) > k*min_dif, the system sets end of segment to n + 1, stores P(b) and sets P(b) = n - j and vm(b) = previous_vm(b) (step S90). Then, the system ranks acc_dif(j), ... acc_dif(n-1) values in increasing order, stores rank value associated with every video frame and sets length of segment P(b) = n-j (step S91). In step S92, if Largest rank value > ρmax ,the system sets ρmax = largest rank value (step S93). Next, setting b = b + 1 and j = n (step S94), the process returns to step S73. In step S88, if not, the system sets n = n + 1 and previous_vm(b) = vm(b) (step S89) and the process returns to step S76.

[0045] Following the step S77, the system sets r = 1 (step S95) and starts displaying sequences of frames having rank greater or equal to r (step S96). Then, selling b = 1 (step S97) and p = 1 (step S98), if rank of image p in segment number b is not greater or equal to r (step S99), the system displays video frame (step S100). Next, if P = P(b) (step S101) , b = L (step S103) and r = ρmax (step S105), the system stops the process of the method. In step S101, if P is not equal to P(b), the system sets p = p + 1 (step S102) and returns to step S99. In step S103, if b is not equal to L,

the system sets b = b + 1 (step S104) and returns to step S98. In step S105, r is not equal to ρmax, the system sets r = r + 1 (step S106) and returns to step S96.

[0046] A video skimming system can be constructed to display a skimmed video summary of frames of any specified range of ranks. This effectively provides temporal compression of the original video sequence. For this purpose, the distortion rank of each of the video frames must be stored so the display system can identify the video frames to be included in the designated summary. Index and rank information can be stored in a database with the video sequence to provide temporal, hierarchical descriptions of a video sequence.

[0047] A system displaying hierarchical summaries is an alternative to showing video frames following a linear "low temporal frequency" to "high temporal frequency" decomposition. In each segment, the rank 1 output of the vector rank filter can be viewed as the "average" key frame since it is the most representative frame of all frames of the segment. Conversely, the video frame of highest rank can be viewed as the frame displaying the most content detail since it is the least representative frame of the segment. The system is nonlinear with the benefit that temporal filtering is not required. Table B illustrates a computer program in the Java source language for generating hierarchical video summaries. In addition to performing automatic segment boundary detection, the program applies the vector median process of embodiment 1 to the keyframes belonging to the same shot. In effect the Java source code provided in Table B provides a way to generate a second, coarser summary of the video sequence, where only one keyframe is generated per video shot. Like in the source code of Table A, the fine and coarse video summaries are calculated on-line.

[0048] In connection with identification of key frames utilizing the techniques of this invention, hierarchical summaries of key frames can be generated utilizing other methods, including a Linde-Buzo-Gray clustering algorithm (also known as a K-means algorithm) as described in the paper AN ALGORITHM FOR VECTOR QUANTIZER DESIGN, IEEE Transactions on Communications, January 1980; or a method such as that described in the co-pending application of Ratakonda, Serial No. 08/994,558, filed December 19, 1997, which relies on the rate of change of images in consecutive frames to constrain the clustering process.

[0049] FIG. 8 illustrates a summarizing appliance 80 for constructing summaries of video sequences according to the methods of the three embodiments of the present invention. A video sequence comprising a plurality of video fields or frames 82 is input to a receiver 84 of the appliance. The frames 82 or some representations thereof are transferred from the receiver 84 to a vector convertor 86 which characterizes each frame as a feature vector. As successive feature vectors representing successive frames 82 of the video sequence are defined, the feature vectors are accumulated in an accumulator 88 in an expanding set of vectors. The distortions or cumulative distance measures of the set of vectors are calculated in the vector filter 90 as new feature vectors are added to the set The vector filter 90 ranks the feature vectors according to the distortion that each feature vector produces in the set of feature vectors in the accumulator 88. If video segments have not been identified in the input frames 82 the cumulative distance associated with the most recent vector added to the set is calculated in the vector filter 90 and compared to some threshold value in a comparator 92. When the cumulative distance associated with the feature vector most recently added to the set has obtained some predefined relationship to the threshold value, a segment boundary is declared. When a segment boundary is declared in the comparator 92 or if segment boundaries are defined in the input frames 82, the output of the vector filter 90 is used in the skimmer 94 to identify and record the frames that are most representative of the content of the video sequence. If the vector filter 90 is based on the more specialized vector median filter, the skimmer 94 can identify the most representative frame as a key frame 96 in each video segment. A vector filter 90 based on the more general vector rank filter may be used to rank frames according to the relative homogeneity of their content or their representativeness of the content of a video segment The video skimmer 94 can identify frames according to a hierarchy of representativeness based on ranking the feature vectors according to relative cumulative distance or according to a clustering technique. With this technique, multiple key frames 96 can be selected from each segment and hierarchical summaries of the plurality of frames 82 can be created.

[0050] The terms and expressions that have been employed in the foregoing specification are used as terms of description and not of limitation, and there is no intention, in the use of such terms and expressions, of excluding equivalents of the features shown and described or portions thereof, it being recognized that the scope of the invention is defined and limited only by the claims that follow.

TABLE A

```
/***********************************************************************************************
*/

/** Java applet implementing on-line video summarization      **/
/** using the vector median filter                            **/

import java.awt.*;
import java.applet.*;
import java.net.*;
import java.awt.image.*;

public class vm_applet extends Applet implements Runnable
{

        private final int OX = 260;
        private final int OY = 0;
        private final int REFRESH = 100;
        private final int LOOP_WAIT = 4999;
        private final int down_off = 145;
        private final int left_off = 0-250;
        private final int HISTLEN = 256;
        private final int MAX_KEYFRAMES = 5;

        private final double alpha = 3.5;
        private final int minShotLength = 10;
        private final int seg_maxlen = 45;
        private final int seg_thresh = 17;
        private final int min_runlen = 7;

        private static int xPos = 40;
        private static int yPos = 40;
        private static int pbest_init;

        private String [] msgs;
        private Image [] imgs;
        private Image [] KeyFrame_imgs;
        private int img_count = 0;
        Graphics currg = null;
        ImageObserver io;
        private static int paint_flag;
        private static String str;
        private int width;
        private int height;
        static int loaded_count = 0;
```

```
static boolean imgs_all_loaded = false;
private int [][] hist;
private int [] store_lowseg;
private int [] store_hiseg;
private int [] acc_dif;
private int count_segs = 0;
private boolean [] flag_shot;
private MemoryImageSource [] theKeyFrames;
private int ScanBase;
private Image theLightGrayImage;


private double mu = 0.0;
private double sigma = 0.0;
private int counter = 0;
private int count_vm = 0;
private int prev_count_vm = 0;
private int seg_len = 0;
private int vm_index=0;
private int prev_vm_index=0;
private int ScanKeyFrames = 0;
private int prev = 0;
private int base = 0;

Thread t = null;

public void init(){

        String num_images;


        num_images = getParameter("ImageCount");
        img_count = Integer.valueOf(num_images).intValue();
        msgs = new String[img_count];

        for( int k = 0; k < img_count; k++)
        {
                msgs[k] = getParameter("Image"+String.valueOf(k));
        }

        // Prepare screen display environment
        Rectangle ctn = getParent().bounds();
        setBackground(Color.lightGray);
        resize(ctn.width, ctn.height);
        currg = getGraphics();
```

```
            imgs = new Image[img_count];
            KeyFrame_imgs = new Image[MAX_KEYFRAMES];

            hist  = new int[img_count][];
            store_lowseg = new int[img_count];
            store_hiseg = new int[ img_count];
            acc_dif = new int[ img_count ];
            flag_shot = new boolean [img_count];


            // Provision memory for histograms
            for( int k = 0; k < img_count; k++)
            {
                hist[k] = new int[HISTLEN];
            }

            theKeyFrames = new MemoryImageSource[img_count];
            theLightGrayImage = MakeLightGrayImage(90,60);

            // HISTLEN  entries in histogram
            // 256 is max sample value difference
            pbest_init = img_count * HISTLEN * 256;
        }


        // Utilities for displaying video sequence and
        // keyframes

        public void paint(Graphics g, int xcoord, int ycoord, Image frame, int fwidth,
int fheight, String caption)
        {


            if( g == null )
            {
                g = getGraphics();
            }
            g.setFont( new Font("Arial", Font.BOLD, 11) );
            io = null;


            g.drawImage(frame,xcoord,ycoord,Color.black,io);
            g.setColor(Color.yellow);
            g.fillRect(xcoord,ycoord+fheight,fwidth,15);
            g.setColor(Color.blue);
```

```
                g.drawString(caption,xcoord,ycoord+fheight+10);

        }

        public void paint(Graphics g, int xcoord, int ycoord, Image [] keyframe, int
kfwidth, int kfheight, int entryBase, int KeyFrameCount, int [] lowBound, int []
highBound, boolean [] shot_detect )
        {

                int index;
                int shiftx;

                if( g == null )
                {
                        g = getGraphics();
                }
                g.setFont( new Font("Arial", Font.BOLD, 11) );
                io = null;


                shiftx = 0;
                index = 0;

                for( int kf = entryBase; kf < entryBase+MAX_KEYFRAMES; kf++)
                {

g.drawImage(keyframe[index++],xcoord+shiftx,ycoord,Color.black,io);

                        if( kf  < KeyFrameCount )
                        {
                                str = " #" + String.valueOf( lowBound[kf] );
                                str += " of [";
                                str += String.valueOf(lowBound[kf]);
                                str += ":";
                                str += String.valueOf(highBound[kf]);
                                str += "]";
                                if( shot_detect[kf] == true )
                                {
                                        g.setColor(Color.green);
                                }
                                else
                                {
                                        g.setColor(Color.yellow);
```

```
                }
                g.fillRect(xcoord+shiftx,ycoord+kfheight,kfwidth,15);
                g.setColor(Color.red);
                g.drawString(str,xcoord+shiftx,ycoord+kfheight+10);
            }
            else
            {
                g.setColor(Color.lightGray);
                g.fillRect(xcoord+shiftx,ycoord+kfheight,kfwidth,15);
            }

            shiftx += (kfwidth+10);
        }
    }

    public void paint(Graphics g)
    {

        g.setColor(Color.red);
        g.setFont( new Font("Arial", Font.BOLD, 11) );
        switch( paint_flag )
        {
            case 0:
                g.drawString(null,0,0);
                break;

            case 1:
                g.drawString(str,xPos,yPos );
                break;

            default:
        }
    }


    public void start()
    {
        if( t == null )
        {
            t = new Thread(this);
            t.start();
        }
    }
    public void run()
    {
```

```
int scan_imgs;
boolean new_keyframe_detected;
Image [] ref_images;
Image [] ref_keyframes;
int ref_width;
int ref_height;
int ref_keyframeCount;
int [] ref_segmentLowBoundaries;
int [] ref_segmentHighBoundaries;
boolean [] ref_shotBoundariesIndicator;
int ref_displayedKeyframeIndexBase;
int ref_seqlen;

// Get image references
paint_flag = 1;
for( int k =0; k < img_count; k++)
{
        try
        {
                imgs[k] = Toolkit.getDefaultToolkit().getImage( new URL(
getDocumentBase(), msgs[k] ));

                str = "Getting Reference to image ";
                str += String.valueOf(k);
                t.sleep( 10 );
                repaint();
        }
        catch( Exception e )
        {
        }
}

// Load images
paint_flag = 0;
repaint();
for( int k = 0; k < img_count; k++)
{
        try
        {
                io = this;
                currg.drawImage(imgs[k],-1000,-1000,Color.black,io);
                t.sleep( 100 );
        }
        catch( Exception e )
```

```
                {
                }
        }

        while( !imgs_all_loaded )
        {
                try
                {
                        str = "Loading images... ";
                        paint_flag = 1;
                        xPos = 40;
                        yPos = 40;
                        repaint();
                }
                catch( Exception  e)
                {
                }
        }

        // Start video summarization

        while( true )
        {
                try
                {
                        str = "7.4925 Hz Progressive 180x120 Video";
                        paint_flag = 1;
                        xPos = OX + 220 ;
                        yPos = OY + 80;
                        repaint();
                }
                catch( Exception e )
                {
                }


                ref_seqlen = getSequenceLength();

                // scan_imgs represents the number of video frames
                // in the set
                for( scan_imgs = 0; scan_imgs < ref_seqlen; scan_imgs++)
                {

                        // summarize and see if segment boundary
                        // has been detected
```

```
                    new_keyframe_detected = summarize( scan_imgs );

                    // Get references to arrays to prepare
                    // display of video sequence and keyframes

                    ref_images = getImages();
                    ref_keyframes = getKeyframes();
                    ref_width = getWidth();
                    ref_height = getHeight();
                    ref_keyframeCount = getKeyframeCount();

                    ref_segmentLowBoundaries =
getSegmentLowBoundaries();
                    ref_segmentHighBoundaries =
getSegmentHighBoundaries();
                    ref_shotBoundariesIndicator =
getShotBoundariesIndicator();
                    ref_displayedKeyframeIndexBase =
getDisplayedKeyframeIndexBase();

                    // display
                    try
                    {
                        paint(currg, OX, OY, ref_images[scan_imgs],
ref_width, ref_height, msgs[scan_imgs]);

                        if( new_keyframe_detected == true )
                        {
                            paint(currg, OX+left_off, OY+down_off,
ref_keyframes, ref_width>>1, ref_height>>1, ref_displayedKeyframeIndexBase,
ref_keyframeCount, ref_segmentLowBoundaries, ref_segmentHighBoundaries,
ref_shotBoundariesIndicator );

                        }
                        else
                        {
                            t.sleep( REFRESH);
                        }
                    }

                    catch( Exception e)
                    {
                    }

                }
```

```
                try
                {
                        t.sleep( LOOP_WAIT );
                }
                catch( Exception e)
                {
                }

        }
}

public boolean summarize( int glob_j )
{

        int diftot;
        int difval;
        double currentMean;
        double threshold;
        int pbest;
        int frac_num;
        int frac_deno;
        boolean keyframe_flag;

        keyframe_flag = false;
        width = 0;
        while( (width = imgs[glob_j].getWidth(null)) == 0)
        {
        }
        height = 0;
        while( (height = imgs[glob_j].getHeight(null)) == 0)
        {
        }

        // calculate histogram of new video frame
        GetHist(0,0,width,height,glob_j);

        if( glob_j == 0 )
        {

                // initialize cumulative distance registers
                for( int k = 0; k < img_count; k++)
                {
                        acc_dif[k] = 0;
                        flag_shot[k] = false;
                }
```

```
                mu = 0.0;
                sigma = 0.0;
                counter = 1;
                prev = 0;
                prev_count_vm = 0;
                count_vm = 0;
                count_segs = 0;
                seg_len = 0;
                vm_index = 0;
                prev_vm_index = 0;

                ScanKeyFrames = 0;
                ScanBase = 0;
                base = 0;

                for( int imgindex = 0; imgindex < MAX_KEYFRAMES;
imgindex++)
                {
                        KeyFrame_imgs[imgindex] = theLightGrayImage;
                }

                flag_shot[0] = true;
                return( keyframe_flag );
        }

        // Shot boundary detection
        diftot = 0;
        for( int m = 0; m < HISTLEN; m++)
        {
                difval = hist[glob_j][m] - hist[glob_j-1][m];
                if( difval > 0 )
                {
                        diftot += difval;
                }
                else
                {
                        diftot -= difval;
                }
        }
        mu += diftot;
        sigma += (diftot * diftot);
        currentMean = mu / (double) counter;
        threshold = currentMean;
        if( counter > 1 )
        {
```

```
                    threshold += (alpha * Math.sqrt( (sigma -
((double)counter*currentMean*currentMean)) / (double)(counter-1) ));
            }

            // This test determines whether a shot boundary is present
            if( (diftot > threshold) && (glob_j-prev > minShotLength) )
            {
                    // Shot boundary has been detected.
                    // A keyframe will be generated
                    prev = glob_j;
                    mu = 0.0;
                    sigma = 0.0;
                    counter = 0;
                    flag_shot[count_segs+1] = true;


                    GetDecim(0,0,width,height,base,count_segs);
                    store_lowseg[count_segs ] = base;
                    store_hiseg[count_segs++] = glob_j-1;
                    base = glob_j;
                    seg_len = 0;
                    prev_vm_index = base;
                    acc_dif[base] = 0;
                    count_vm = 0;

                    keyframe_flag = true;

            }
            else
            {
                    counter++;

                    // calculate vector median filter
                    acc_dif[ glob_j ] = 0;
                    pbest = pbest_init;
                    // calculate cumulative distances
                    for( int b = base; b < glob_j; b++)
                    {
                            diftot = 0;
                            for( int m = 0; m < HISTLEN; m++)
                            {
                                    difval = hist[b][m] - hist[glob_j][m];
                                    if( difval > 0 )
                                    {
                                            diftot += difval;
```

```
                        }
                        else
                        {
                                diftot -= difval;
                        }
                }
                // Calculate cumulative distance for newest
                // feature vector
                acc_diff[ glob_j ] += diftot;
                // Update cumulative distance for the feature
                // vector in the set
                acc_diff[ b ] += diftot;
                // Keep track of minimum cumulative distance
                if( acc_diff[ b ] < pbest )
                {
                        vm_index = b;
                        pbest = acc_diff[ b ];
                }
        }
        // Keep track of minimum cumulative distance
        if( acc_diff[ glob_j ] < pbest )
        {
                vm_index = glob_j;
        }
        // keep track of how many times the same
        // video frame has been selected consecutively by the
        // vector median filter.

        prev_count_vm = count_vm;
        if( vm_index == prev_vm_index )
        {
                count_vm++;
        }
        else
        {
                count_vm = 0;
        }

        // Calculate threshold value
        // frac_num is numerator and frac_deno is denoinator
        frac_num = seg_maxlen*(1+seg_thresh);
        frac_deno = seg_maxlen+(seg_thresh*seg_len);

        // Detect a segment boundary and issue a keyframe
// if minimum run length has been exceeded
```

```
                    if( (prev_count_vm > min_runlen) && (count_vm == 0))
                    {
                        GetDecim(0,0,width,height,base,count_segs);
                        store_lowseg[count_segs ] = base;
                        store_hiseg[count_segs++] = glob_j-1;

                        base = glob_j;
                        seg_len = 0;
                        prev_vm_index = base;
                        acc_dif[base] = 0;

                        keyframe_flag = true;

                    // Detect a segment boundary and issue a keyframe
// if cumulative distance
                    // of newest feature vector is larger than the
                    // thershold value multiplied by the minimum
                    // cumulative distance of the set.
                    else if( (frac_deno*acc_dif[glob_j]) >=
(frac_num*acc_dif[vm_index]) )
                    {
                        GetDecim(0,0,width,height,base,count_segs);
                        store_lowseg[count_segs ] = base;
                        store_hiseg[count_segs++] = glob_j-1;
                        base = glob_j;
                        seg_len = 0;
                        prev_vm_index = base;
                        acc_dif[base] = 0;
                        count_vm = 0;

                        keyframe_flag = true;
                    }
                    // No segment boundary detected;
                    else
                    {
                        seg_len++;
                        keyframe_flag = false;
                        prev_vm_index = vm_index;
                    }
                // Prepare display of the keyframe
                // only MAX_KEYFRAMES are displayed
                // on the screen
                if( keyframe_flag == true )
                {
                    if( ScanKeyFrames == MAX_KEYFRAMES )
```

```
                    {
                            // Place new  keyframe at beginning
                            KeyFrame_imgs[0] = createImage(
theKeyFrames[count_segs-1] );
                            // and make other placeholders gray
                            for( int imgindex = 1; imgindex < MAX_KEYFRAMES;
imgindex++)
                            {
                                    KeyFrame_imgs[imgindex] = theLightGrayImage;
                            }
                            ScanBase = count_segs-1;
                            ScanKeyFrames = 1;
                    }
                    else
                    {
                            // append new keyframe to display list
                            KeyFrame_imgs[ScanKeyFrames++] = createImage(
theKeyFrames[count_segs-1] );
                    }
            }

            // Take care of the last keyframe of the last segment
            // in the video sequence
            if( glob_j == img_count-1 )
            {
                    if( keyframe_flag == false )
                    {
                            GetDecim(0,0,width,height,base,count_segs);
                            store_lowseg[count_segs ] = base;
                            store_hiseg[count_segs++] = glob_j;

                            keyframe_flag = true;
                    }
                    else
                    {
                            GetDecim(0,0,width,height,glob_j,count_segs);
                            store_lowseg[count_segs ] = glob_j;
                            store_hiseg[count_segs++] = glob_j;
                    }

                    if( ScanKeyFrames == MAX_KEYFRAMES )
                    {
                            KeyFrame_imgs[0] = createImage(
theKeyFrames[count_segs-1] );
```

```
                    for( int imgindex = 1; imgindex < MAX_KEYFRAMES;
imgindex++)
                    {
                         KeyFrame_imgs[imgindex] = theLightGrayImage;
                    }
                    ScanKeyFrames = 1;
                    ScanBase = count_segs-1;
               }
               else
               {
                    KeyFrame_imgs[ScanKeyFrames++] = createImage(
theKeyFrames[count_segs-1] );
               }

          }

          return( keyframe_flag );
     }

     // Utility for loading images

     public boolean imageUpdate(Image img, int flags, int x, int y, int w, int h)
     {

          if( (flags & (ERROR | ABORT)) != 0 )
          {
               paint_flag = 1;
               str = "ERROR IN LOADING";
               xPos = 20;
               yPos = 10;
               repaint();
          }

          if( imgs_all_loaded )
          {
               return false;
          }
          if( (flags & ALLBITS) == 0 )
          {
               return true;
          }

          if( ++loaded_count == img_count )
          {
               imgs_all_loaded = true;
```

```
        }
        return false;
    }

    // Utility for computing an image histogram
    private void computeHistogram( int [] pixels, int w, int h, int n)
    {

        int pixval;
        int r;
        int g;
        int b;
        int lum;
        int scan_index = 0;

        for( int k = 0; k < HISTLEN; k++)
        {
            hist[n][k] = 0;
        }

        for (int j=0; j<h; j++)
        {
            for (int i=0; i<w; i++)
            {
                pixval = pixels[scan_index++];
                r = (pixval >> 16) & 0xff;
                g = (pixval >> 8) & 0xff;
                b = (pixval    ) & 0xff;
```

```
                    lum = (int)((0.299*r)+(0.587*g)+(0.114*b));
                    hist[n][lum]++;

                }
            }
            return;
        }

        // Utility for down-sampling an image
        private void computeDecim( int [] pixels, int w, int h, int [] decim)
        {

            int scan_index = 0;
            int decim_index = 0;

            for  (int  j=0;  j<h;  j+=2)
            {
                for  (int  i=0;  i<w;  i+=2)
                {
                    decim[decim_index++] = pixels[scan_index];
                    scan_index += 2;
                }
                scan_index += w;
            }
            return;
        }

        private void  GetHist(int x, int y, int w, int h, int n)
        {
            int[]  pixels  =  new  int[w*h];

        PixelGrabber  pg  =
            new  PixelGrabber(imgs[n], x, y, w, h, pixels, 0, w);
        try
            {
            pg.grabPixels();
        }
            catch  (InterruptedException  e)
            {
            System.err.println("interrupted  waiting  for  pixels!");
            return;
```

```
        }
        if ((pg.status() & ImageObserver.ABORT) != 0)
            {
                System.err.println("image fetch aborted or errored");
                return;
        }
            computeHistogram(pixels,w,h,n);

        }

        private void GetDecim(int x, int y, int w, int h, int n, int u)
        {
            int[] pixels = new int[w*h];
            int[] decim_pix = new int[(w*h)>>2];

        PixelGrabber pg =
            new PixelGrabber(imgs[n], x, y, w, h, pixels, 0, w);
        try
            {
            pg.grabPixels();
        }
            catch (InterruptedException e)
            {
            System.err.println("interrupted waiting for pixels!");
            return;
        }

        if ((pg.status() & ImageObserver.ABORT) != 0)
            {
                System.err.println("image fetch aborted or errored");
                return;
        }
            computeDecim(pixels,w,h,decim_pix);
            theKeyFrames[u] = new
MemoryImageSource(w>>1,h>>1,decim_pix,0,w>>1);
            return;
        }

        private Image MakeLightGrayImage(int w, int h)
        {
            int[] pixels = new int[w*h];

            for (int j=0; j<(w*h); j++)
            {
```

```
                pixels[j] = 0xFFB8B8B8;
            }
            return( createImage( new MemoryImageSource(w,h,pixels,0,w)) );
    }

    public Image [] getImages()
    {
            return( imgs );
    }

    public Image [] getKeyframes()
    {
            return( KeyFrame_imgs );
    }
    public int getWidth()
    {
            return( width );
    }
    public int getHeight()
    {
            return( height );
    }
    public int getKeyframeCount()
    {
            return( count_segs );
    }
    public int [] getSegmentLowBoundaries()
    {
            return( store_lowseg );
    }
    public int [] getSegmentHighBoundaries()
    {
            return( store_hiseg );
    }
    public boolean [] getShotBoundariesIndicator()
    {
            return( flag_shot );
    }
    public int getDisplayedKeyframeIndexBase()
    {
            return( ScanBase );
    }
    public int getSequenceLength()
    {
            return( img_count );
    }
```

TABLE B

```
/********************************************************************************************
**/

/** On-line video summarization based on vector median filter **/
/** A second, coarser summary is also generated             **/
/** by applying the vector median filter to the keyframes   **/
/** belonging to a same shot                                     **/

import java.awt.*;
import java.applet.*;
import java.net.*;
import java.awt.image.*;

public class vm_applet extends Applet implements Runnable
{

        static final int OX = 260;
        static final int OY = 0;
        static final int REFRESH = 0;
        static final int LOOP_WAIT = 4999;
        static final int down_off = 145;
        static final int left_off = 0-255;
        static final int HISTLEN = 256;

        static int xPos = 40;
        static int yPos = 40;
        String num_images;
        String [] msgs;
        static Image [] imgs;
        static Image [] proc_imgs;
        int img_count = 0;
        Graphics currg = null;
        ImageObserver io;
        static int paint_flag;
        static String str;
        static int load_index = 0;
        int width;
        int height;
        static int loaded_count = 0;
        static boolean imgs_all_loaded = false;
        static int [] save_xPos;
        static int [] save_yPos;
        int [][] hist;
```

```
static int glob_j;
static int [] store_lowseg;
static int [] store_hiseg;
static int [] store_index;
static int [] acc_dif;
static int [] super_acc;
int vm_index;
int prev_vm_index;
int shiftx = 0;
int shifty = 0;
int count_segs = 0;
boolean keyframe_flag;
static boolean [] flag_shot;
int key_count;
static int pbest_init;
int super_base;
int count_supers;
static int [] super_vm_index;

int dic;

Thread t = null;

public void init(){

        num_images = getParameter("ImageCount");
        img_count = Integer.valueOf(num_images).intValue();
        msgs = new String[img_count];

        for( int k = 0; k < img_count; k++)
        {
                msgs[k] = getParameter("Image"+String.valueOf(k));
        }

        // Provision memory for various parameters

        Rectangle ctn = getParent().bounds();
        setBackground(Color.lightGray);
        resize(ctn.width, ctn.height);
        currg = getGraphics();
        imgs = new Image[img_count];
        proc_imgs = new Image[img_count];
        save_xPos = new int[img_count];
        save_yPos = new int[img_count];
```

```
            hist  = new int[img_count][];
            store_lowseg = new int[img_count];
            store_hiseg = new int[ img_count];
            acc_dif = new int[ img_count ];
            super_acc = new int[ img_count ];
            super_vm_index = new int[ img_count ];
            store_index = new int[img_count];
            flag_shot = new boolean [img_count];

            for( int k = 0; k < img_count; k++)
            {
                  hist[k] = new int[HISTLEN];
            }

            // 256 entries in histogram
            // 256 is max sample value difference
            pbest_init = img_count * HISTLEN * 256;
      }

      // Utilities for displaying images and text on screen
      // Text (frame number associated with keyframe) is
      // in a yellow box if it is a regular keyframe
      // Text is in a green box if keyframe is also first
      // keyframe of a video shot
      // Image is surrounded by a blue frame is keyframe is
      // also keyframe at coarse summary level.

      public void paint(Graphics g)
      {

            if( currg == null )
            {
                  currg =g;
            }

            g.setColor(Color.red);
            g.setFont( new Font("Arial", Font.BOLD, 11) );
            switch( paint_flag )
            {
                  case 0:
                        g.drawString(null,0,0);
                        break;
                  case 1:
                        g.drawString(str,xPos,yPos );
                        break;
```

```
                        case 2:
                                io = null;
                                for( int idx = 0; idx <= load_index; idx++)
                                {

g.drawImage(imgs[idx],save_xPos[idx],save_yPos[idx],Color.black,io);

g.drawString(msgs[idx],save_xPos[idx],save_yPos[idx]+120);
                                }
                                load_index += 1;
                                break;
                        case 3:
                                io = null;
                                break;
                        case 4:
                                io = null;
                                break;

                        case 5:
                                io = null;
                                g.drawImage(imgs[glob_j],xPos,yPos,Color.black,io);
                                g.setColor(Color.yellow);
                                g.fillRect(xPos,yPos+height,width,15);
                                g.setColor(Color.red);
                                g.drawString(msgs[glob_j],xPos,yPos+height+10);
                                break;

                        case 6:
                                io = null;

                                shiftx = 0;
                                shifty = 0;
                                key_count = 0;
                                int ss = 0;
                                for( int kf = 0; kf < count_segs; kf++)
                                {
                                        g.drawImage(proc_imgs[kf],xPos+left_off+shiftx,y
Pos+shifty+down_off,Color.black,io);

                                        str = " #" + String.valueOf( store_index[kf] );
                                        str += " of [";
                                        str += String.valueOf(store_lowseg[kf]);
                                        str += ":";
                                        str += String.valueOf(store_hiseg[kf]);
                                        str += "]";
```

```
                        if( flag_shot[kf] == true )
                        {
                                g.setColor(Color.green);
                        }
                        else
                        {
                                g.setColor(Color.yellow);
                        }
g.fillRect(xPos+left_off+shiftx,yPos+down_off+shifty+(height>>1),(width>>1),15);
                        g.setColor(Color.red);
g.drawString(str,xPos+left_off+shiftx,yPos+shifty+down_off+(height>>1)+10);

                        if( (ss < count_supers) && (kf ==
super_vm_index[ss]) )
                        {
                                g.setColor(Color.blue);
g.fillRect(xPos+left_off+shiftx,yPos+down_off+15+shifty+(height>>1),(width>>1),2);
g.fillRect(xPos+left_off+shiftx,yPos+down_off+shifty-2,(width>>1),2);
g.fillRect(xPos+left_off+shiftx-2,yPos+down_off+shifty-2,2,(height>>1)+15+4);
                                g.fillRect(xPos+left_off+shiftx+(width>>1),y
Pos+down_off+shifty-2,2,(height>>1)+15+4);
                                ss++;
                        }

                        shiftx += (width >> 1)+10;
                        if( ++key_count >= 7 )
                        {
                                key_count = 0;
                                shiftx = 0;
                                shifty += ((height>>1)+25);
                        }

                }

                break;
```

```
                default:
            }
        }

        public void start()
        {
            if( t == null )
            {
                t = new Thread(this);
                t.start();
            }
        }
        public void run()
        {
            int diftot;
            int difval;
            int base;
            double mu ;
            double sigma;
            int counter ;
            double alpha = 4.0;
            double minShotLength = 10;
            double currentMean;
            int prev;
            double threshold;
            int pbest;
            int count_vm;
            int prev_count_vm;
            // Maximum segment length
            // At least one keyframe will be generated
            // for that many input video frames
            int seg_maxlen = 45;
            // Initial value for calculating time-varying
            // threshold.
            int seg_thresh = 17;
            // Minimum number of times the same frame
            // must be consecutively selected by
            // median filter to become a keyframe
            int min_runlen = 7;
            int seg_len;
            double frac_num;
            double frac_deno;
            int super_pbest;
            int index;
```

```
                // get references to images from HTML document
                // publishing this applet
                paint_flag = 1;
                for( int k =0; k < img_count; k++)
                {
                        try
                        {
                                imgs[k] = Toolkit.getDefaultToolkit().getImage( new URL(
getDocumentBase(), msgs[k] ));

                                str = "Getting Reference to image ";
                                str += String.valueOf(k);
                                repaint();
                                // Thread.currentThread().sleep(100);
                        }
                        catch( Exception e )
                        {
                        }
                }
                paint_flag = 0;
                repaint();
                for( int k = 0; k < img_count; k++)
                {
                        io = this;
                        currg.drawImage(imgs[k],-1000,-1000,Color.black,io);
                        try
                        {
                                Thread.sleep( 100 );
                        }
                        catch( Exception e )
                        {
                        }
                }

                while( !imgs_all_loaded )
                {
                        try
                        {
                                str = "Loading images... ";
                                paint_flag = 1;
                                xPos = 40;
                                yPos = 40;
                                repaint();
                                // Thread.sleep(10);
                        }
```

```
            catch( Exception  e)
            {
            }
        }

        while( true )
        {
            try
            {
                str = "7.4925 Hz Progressive 180x120 Video";
                paint_flag = 1;
                xPos = OX + 220 ;
                yPos = OY + 80;
                repaint();
                // Thread.sleep(2000);
            }
            catch( Exception e )
            {
            }

            for( int k = 0; k < img_count; k++)
            {
                acc_dif[k] = 0;
                store_index[k] = 0;
                flag_shot[k] = false;
            }

            mu = 0.0;
            sigma = 0.0;
            counter = 1;
            prev = 0;
            prev_count_vm = 0;
            count_vm = 0;
            count_segs = 0;
            seg_len = 0;
            vm_index = 0;
            super_base = 0;
            count_supers = 0;

            // Start video summarization
            base = 0;
            for( glob_j = 0; glob_j < img_count; glob_j++)
            {
                try
                {
```

```
width = 0;
while( (width = imgs[glob_j].getWidth(null)) == 0)
{
        // Thread.sleep(1);
}
height = 0;
while( (height = imgs[glob_j].getHeight(null)) == 0)
{
        // Thread.sleep(1);
}

// Compute video frame histogram
GetHist(0,0,width,height,glob_j);

// Initialization for coarse summary

                if( glob_j == 0 )
{
        xPos = OX;
        yPos = OY;
        paint_flag = 5;
        paint(currg);
        Thread.sleep( REFRESH );
        prev_vm_index = 0;
        flag_shot[0] = true;

        super_base = 0;
        count_supers = 0;
        for( int z = 0; z < img_count; z++)
        {
                super_acc[z] = 0;
        }

        continue;
}

// For shot boundary detection
diftot = 0;
for( int m = 0; m < HISTLEN; m++)
{
        difval = hist[glob_j][m] - hist[glob_j-1][m];
        if( difval > 0 )
        {
                diftot += difval;
        }
```

```
                        else
                        {
                                diftot -= difval;
                        }
                }
                mu += diftot;
                sigma += (diftot * diftot);
                currentMean = mu / (double) counter;
                threshold = currentMean;
                if( counter > 1 )
                {
                        threshold += (alpha * Math.sqrt( (sigma -
((double)counter*currentMean*currentMean)) / (double)(counter-1) ));
                }

                // Shot boundary has been detected if
                // following test is true
                if( (diftot > threshold) && (glob_j-prev >
minShotLength) )
                {
                        prev = glob_j;
                        mu = 0.0;
                        sigma = 0.0;
                        counter = 0;
                        flag_shot[count_segs+1] = true;


        GetDecim(0,0,width,height,prev_vm_index,count_segs);
                        store_lowseg[count_segs ] = base;
                        store_hiseg[count_segs] = glob_j-1;
                        store_index[count_segs] = prev_vm_index;
                        base = glob_j;
                        seg_len = 0;
                        prev_vm_index = base;
                        acc_dif[base] = 0;
                        count_vm = 0;

                        // dic = 1;
                        keyframe_flag = true;

            // Since shot boundary has been found,
            // it is time to calculate most representative
            // keyframe of all keyframes in the same shot
            // The resulting keyframe is the keyframe
            // that will appear in the coarser summary
```

```
// (one keyframe per video shot)
// The vector median filter is applied to
// all keyframes in the shot.

        // First calculate cumulative distances
        for( int z = super_base; z < count_segs;
z++)
        {
                index = store_index[z];
                diftot = 0;
                for( int m = 0; m < HISTLEN; m++)
                {
                        difval = hist[index][m] -
hist[prev_vm_index][m];
                        if( difval > 0 )
                        {
                                diftot += difval;
                        }
                        else
                        {
                                diftot -= difval;
                        }
                }
                super_acc[ count_segs ] += diftot;
                super_acc[ z ] += diftot;
        }

        // Then identify minimum cumulative
distance
        super_pbest = pbest_init;
```

```
                    for( int z = super_base; z <= count_segs;
z++)
                    {
                        if( super_acc[ z ] < super_pbest )
                        {
super_vm_index[count_supers] = z;
                            super_pbest = super_acc[ z ];
                        }
                    }
                    super_base = count_segs;
                    count_supers++;
                    count_segs++;
                }
                else
                {
                    // No shot boundary has been detected
                    counter++;

                    // calculate output of vector median
                    // filter for current segment
                    acc_dif[ glob_j ] = 0;
                    pbest = pbest_init;
                    for( int b = base; b < glob_j; b++)
                    {
                        diftot = 0;
                        for( int m = 0; m < HISTLEN; m++)
                        {
                            difval = hist[b][m] -
hist[glob_j][m];
                            if( difval > 0 )
                            {
                                diftot += difval;
                            }
                            else
                            {
                                diftot -= difval;
                            }
                        }
                // Update cumulative distances
                        acc_dif[ glob_j ] += diftot;
                        acc_dif[ b ] += diftot;
                // Keep track of minimal cumulative distance
                        if( acc_dif[ b ] < pbest )
                        {
```

```
                    vm_index = b;
                    pbest = acc_diff[ b ];
                }
            }
            // Keep track of minimum cumulative
distance
            if( acc_diff[ glob_j ] < pbest )
            {
                vm_index = glob_j;
            }

            // Keep track of number of times
            // same keyframe has been selected
            // consecutively

            prev_count_vm = count_vm;
            if( vm_index == prev_vm_index )
            {
                count_vm++;
            }
            else
            {
                count_vm = 0;
            }

            // Calculate threshold numerator and
            // denominator
            frac_num = seg_maxlen*(1.0+seg_thresh);
            frac_deno =
seg_maxlen+(seg_thresh*seg_len);

            if( (prev_count_vm > min_runlen) &&
(count_vm == 0))
            {
            // Issue new keyframe as minimum
            // run length for vector median output
            // has been satisfied

GetDecim(0,0,width,height,prev_vm_index,count_segs);
                store_lowseg[count_segs ] = base;
                store_hiseg[count_segs] = glob_j-1;
                store_index[count_segs] =
prev_vm_index;

                base = glob_j;
                seg_len = 0;
```

```
                        prev_vm_index = base;
                        acc_dif[base] = 0;

                        keyframe_flag = true;

                        // Update cumulative distances
                        // among keyframes of the current
                        // shot (for coarse summary)

                        for( int z = super_base; z <
count_segs; z++)
                        {
                                index = store_index[z];
                                diftot = 0;
                                for( int m = 0; m < HISTLEN;
m++)
                                {
                                        difval = hist[index][m] -
hist[prev_vm_index][m];
                                        if( difval > 0 )
                                        {
                                                diftot += difval;
                                        }
                                        else
                                        {
                                                diftot -= difval;
                                        }
                                }
                                super_acc[ count_segs ] +=
diftot;
                                super_acc[ z ] += diftot;
                        }
                        count_segs++;
                }
                else if( (frac_deno*acc_dif[glob_j]) >=
(frac_num*acc_dif[vm_index]) )
                {
                // Cumulative distance is greater
                // than threshold multiplied by
                // minimal cumlative distance:
                // Time to generate a keyframe

GetDecim(0,0,width,height,prev_vm_index,count_segs);
                        store_lowseg[count_segs ] = base;
                        store_hiseg[count_segs] = glob_j-1;
```

```
                store_index[count_segs] = prev_vm_index;

                base = glob_j;
                seg_len = 0;
                prev_vm_index = base;
                acc_dif[base] = 0;
                count_vm = 0;

                keyframe_flag = true;

                // Update cumulative distance
                // among keyrames of current
                // shot (for coarse summary)
                for( int z = super_base; z < count_segs; z++)
                {
                        index = store_index[z];
                        diftot = 0;
                        for( int m = 0; m < HISTLEN; m++)
                        {
                                difval = hist[index][m] - hist[prev_vm_index][m];
                                if( difval > 0 )
                                {
                                        diftot += difval;
                                }
                                else
                                {
                                        diftot -= difval;
                                }
                        }
                        super_acc[ count_segs ] += diftot;
                        super_acc[ z ] += diftot;
                }
                count_segs++;
        }
        else
        {
                // No keyframe is generated
                seg_len++;
                keyframe_flag = false;
                prev_vm_index = vm_index;
                Thread.sleep(100);
```

```
                    }
                }

                xPos = OX;
                yPos = OY;
                paint_flag = 5;
                paint(currg);

                if( keyframe_flag == true )
                {
                    paint_flag =6;
                    paint(currg);
                }
                Thread.sleep( REFRESH );

            }
            catch( Exception e )
            {
            }
        }

        // Take care of last segment in video sequence
        if( keyframe_flag == false )
        {
            GetDecim(0,0,width,height,vm_index, count_segs);
            store_lowseg[count_segs ] = base;
            store_hiseg[count_segs] = glob_j-1;
            store_index[count_segs] = vm_index;
        }
        else
        {
            GetDecim(0,0,width,height,glob_j-1,count_segs);
            store_lowseg[count_segs ] = glob_j-1;
            store_hiseg[count_segs] = glob_j-1;
            store_index[count_segs] = glob_j-1;
        }

        // Take care of coarse summary for last shot
        // in the video sequence
        for( int z = super_base; z < count_segs; z++)
        {
            index = store_index[z];
            diftot = 0;
            for( int m = 0; m < HISTLEN; m++)
            {
```

```
                        difval = hist[index][m] - hist[vm_index][m];
                        if( difval > 0 )
                        {
                                diftot += difval;
                        }
                        else
                        {
                                diftot -= difval;
                        }
                }
                super_acc[ count_segs ] += diftot;
                super_acc[ z ] += diftot;
        }
        super_pbest = pbest_init;
        for( int z = super_base; z <= count_segs; z++)
        {
                if( super_acc[ z ] < super_pbest )
                {
                        super_vm_index[count_supers] = z;
                        super_pbest = super_acc[ z ];
                }
        }
        count_supers++;
        count_segs++;

        try
        {

                xPos = OX;
                yPos = OY;
                paint_flag = 6;
                paint(currg);
                Thread.sleep( LOOP_WAIT );
        }
        catch( Exception e)
        {
        }

        try
        {
                paint_flag = 0;
                paint( currg );
                // Thread.sleep( 10 );
        }
        catch( Exception e )
```

```
                {
                }

            }

        }

        // Utility for loading the images

        public boolean imageUpdate(Image img, int flags, int x, int y, int w, int h)
        {

            if( (flags & (ERROR | ABORT)) != 0 )
            {
                paint_flag = 1;
                str = "ERROR IN LOADING";
                xPos = 20;
                yPos = 10;
                repaint();
            }

            if( imgs_all_loaded )
            {
                return false;
            }
            if( (flags & ALLBITS) == 0 )
            {
                return true;
            }

            if( ++loaded_count == img_count )
            {
                imgs_all_loaded = true;
            }
            return false;
        }

        // Utility for computing an image histogram
        public void computeHistogram( int [] pixels, int w, int h, int n)
        {

            int pixval;
            int r;
            int g;
            int b;
```

```
        int lum;
        int scan_index = 0;

        for( int k = 0; k < HISTLEN; k++)
        {
                hist[n][k] = 0;
        }

        for (int j=0; j<h; j++)
        {
                for (int i=0; i<w; i++)
                {
                        // pixval = pixels[(j*w)+i];
                        pixval = pixels[scan_index++];
                        r = (pixval >> 16) & 0xff;
                        g = (pixval >> 8) & 0xff;
                        b = (pixval     ) & 0xff;

                        lum = (int)((0.299*r)+(0.587*g)+(0.114*b));
                        hist[n][lum]++;

                }
        }
        return;
}

// Utility to compute a down-sampled version of
// an image. This is used to display the keyframes
// on the screen.
public void computeDecim( int [] pixels, int w, int h, int [] decim)
{

        int scan_index = 0;
        int decim_index = 0;

        for (int j=0; j<h; j+=2)
        {
                for (int i=0; i<w; i+=2)
                {
                        decim[decim_index++] = pixels[scan_index];
                        scan_index += 2;
                }
                scan_index += w;
        }
        return;
```

```
}

public void GetHist(int x, int y, int w, int h, int n)
{
        int[] pixels = new int[w*h];

PixelGrabber pg =
        new PixelGrabber(imgs[n], x, y, w, h, pixels, 0, w);
try
        {
        pg.grabPixels();
}
        catch (InterruptedException e)
        {
        System.err.println("interrupted waiting for pixels!");
        return;
}

if ((pg.status() & ImageObserver.ABORT) != 0)
        {
                System.err.println("image fetch aborted or errored");
                return;
}
        computeHistogram(pixels,w,h,n);

}

public void GetDecim(int x, int y, int w, int h, int n, int u)
{
        int[] pixels = new int[w*h];
        int[] decim_pix = new int[(w*h)>>2];

PixelGrabber pg =
        new PixelGrabber(imgs[n], x, y, w, h, pixels, 0, w);
try
        {
        pg.grabPixels();
}
        catch (InterruptedException e)
        {
        System.err.println("interrupted waiting for pixels!");
        return;
}

if ((pg.status() & ImageObserver.ABORT) != 0)
```

```
                {
                        System.err.println("image fetch aborted or errored");
                        return;
        }
                computeDecim(pixels,w,h,decim_pix);
                proc_imgs[u] = createImage(new
    MemoryImageSource(w>>1,h>>1,decim_pix,0,w>>1));
        }

        public boolean mouseDown(Event evt, int x, int y)
        {

                if( load_index < img_count )
                {
                        save_xPos[load_index] = x;
                        save_yPos[load_index] = y;

                        paint_flag = 2;
                        repaint();
                }
                else
                {
                        paint_flag = 2;
                        load_index = img_count-1;
                        repaint();
                }

                return true;
        }

    }
```

Claims

1. A method of creating a summary of a plurality of video frames comprising the steps of:

   (a) receiving a first frame;
   (b) receiving a subsequent frame;
   (c) resolving a homogeneity of content of said first frame and each said subsequent frame received;
   (d) identifying at least one key frame from said first frame and each said subsequent frame received having content most homogeneous with that of said first frame and each said subsequent frame;
   (e) comparing said homogeneity of said first frame and each said subsequent frame with a threshold homogeneity; and
   (f) repeating steps (b) through (e) until said homogeneity has some predefined relationship with said threshold homogeneity.

2. The method of claim 1 wherein said homogeneity of content of said frames is resolved with a vector rank filter (70).

3. A method of identifying a key video frame within a plurality of video frames comprising the steps of:

(a) characterizing said plurality of video frames as a plurality of feature vectors;
(b) identifying a key feature vector that minimizes the distortion of said plurality of feature vectors; and
(c) identifying said video frame corresponding to said key feature vector as said key video frame.

4. The method of claim 3 wherein said key feature vector that minimizes the distortion of said plurality of feature vectors is identified by a vector rank filter (70).

5. A method of identifying a key video frame within a plurality of video frames comprising the steps of:

(a) characterizing said plurality of video frames as a plurality of feature vectors;
(b) identifying a key feature vector characterized by a minimal cumulative distance to all said other feature vectors of said plurality of feature vectors; and
(c) identifying said video frame corresponding to said key feature vector as said key video frame.

6. The method of claim 5 wherein said key feature vector characterized by said minimal cumulative distance is identified by a vector median filter (12).

7. The method of claim 3 or 5 wherein at least one feature vector of said plurality of feature vectors is a vector signal describing an image histogram.

8. The method of claim 3 or 5 wherein at least one feature vector of said plurality of feature vectors is a vector signal describing an ordered set of image samples.

9. The method of claim 3 or 5 wherein at least one feature vector of said plurality of feature vectors is a vector signal which includes a component describing luminance of said video frame.

10. The method of claim 3 or 5 wherein at least one feature vector of said plurality of feature vectors is a vector signal which includes a component describing the average intensity of said video frame.

11. A method of determining a second boundary of a video segment within a plurality of video frames comprising the steps of:

(a) defining a threshold distortion;
(b) locating a first frame in said video segment;
(c) defining a first feature vector representative of said first frame;
(d) including said first feature vector in a set of segment feature vectors;
(e) defining a next feature vector representative of a subsequent video frame;
(f) including said next feature vector with said set of segment feature vectors;
(g) calculating a distortion of said set of segment feature vectors resulting from the inclusion of said next feature vector in said set of segment feature vectors;
(h) comparing said distortion of said set of segment feature vectors with said threshold distortion; and
(i) repeating steps (e) through (h) until said distortion of said set of segment feature vectors has a predefined relationship to threshold distortion thereby defining said second boundary.

12. The method of claim 11 wherein said distortion of said set of segment feature vectors is determined with a vector rank filter (70).

13. The method of claim 11 wherein said threshold distortion has a predefined relationship to a minimum distortion of said set of segment feature vectors.

14. A method of determining a second boundary of a video segment within a plurality of video frames comprising the steps of:

(a) defining a threshold value;
(b) locating a first frame in said video segment;
(c) defining a first feature vector representative of said first frame;

(d) including said first feature vector in a set of segment feature vectors;

(e) defining a next feature vector representative of a subsequent video frame;

(f) including said next feature vector with said set of segment feature vectors;

(g) calculating a cumulative distance from said next feature vector to each said feature vector in said set of segment feature vectors;

(h) updating said cumulative distance;

(i) comparing said updated cumulative distance with said threshold value; and

(j) repeating steps (e) through (i) until said updated cumulative distance has a predefined relationship to said threshold value thereby defining said second boundary.

15. The method of claim 14 wherein a minimum of said cumulative distance is determined with a vector median filter (12).

16. The method of claim 15 wherein said threshold value has a predefined relationship to said minimum cumulative distance of said set of segment feature vectors.

17. The method of claim 13 or 16 wherein said predefined relationship is a function of a noise power of an image in a video frame.

18. The method of claim 13 or 16 wherein said predefined relationship is a function of a compression ratio to be achieved by a summary of said plurality of video frames.

19. The method of claim 11 or 14 wherein said first feature vector and said next feature vector are vector signals describing an image histogram.

20. A method of creating a summary of a plurality of video frames comprising the steps of:

(a) defining a threshold distortion;

(b) locating a first frame in a segment of video frames having relatively homogenous content;

(c) defining a first feature vector representative of said first frame;

(d) including said first feature vector in a set of segment feature vectors;

(e) defining a next feature vector representative of a subsequent video frame;

(f) including said next feature vector with said set of segment feature vectors;

(g) calculating the distortion of said set of segment feature vectors resulting from the inclusion of said next feature vector with said set of segment feature vectors;

(h) comparing said distortion of said set of segment feature vectors with said threshold distortion;

(i) repeating steps (e) through (h) until said distortion of said set of segment feature vectors has a predefined relationship to said threshold distortion thereby defining a second boundary of said video segment;

(j) identifying a key feature vector that minimizes said distortion of said set of segment feature vectors; and

(k) identifying said video frame corresponding to said key feature vector as a key video frame.

21. The method of claim 20 wherein said distortion of said set of segment feature vectors is determined with a vector rank filter (70).

22. The method of claim 20 wherein said threshold distortion has a predefined relationship to a minimum distortion of said set of segment feature vectors.

23. The method of claim 22 wherein said distortion of said set of segment feature vectors is determined with a vector rank filter (70).

24. A method of creating a summary of a plurality of video frames comprising the steps of:

(a) defining a threshold value;

(b) locating a first frame in a segment of video frames having relatively homogenous content;

(c) defining a first feature vector representative of said first frame;

(d) including said first feature vector in a set of segment feature vectors;

(e) defining a next feature vector representative of a subsequent video frame;

(f) including said next feature vector with said set of segment feature vectors;

(g) calculating a cumulative distance from said next feature vector to each said feature vector in said set of segment feature vectors;
(h) updating said cumulative distance;
(i) comparing said updated cumulative distance with said threshold value;
(j) repeating steps (e) through (i) until said updated cumulative distance has a predefined relationship to said threshold value thereby defining said second boundary of said video segment;
(k) identifying a key feature vector that minimizes said cumulative distance; and
(l) identifying said video frame corresponding to said key feature vector as a key video frame.

25. The method of claim 24 wherein a minimum of said cumulative distance is determined with a vector median filter (12).

26. The method of claim 24 wherein said threshold value has a predefined relationship to said minimum cumulative distance of said set of segment feature vectors.

27. The method of claim 26 wherein said minimum cumulative distance is determined with a vector rank filter (70).

28. The method of claim 20 or 24 wherein said first feature vector and said next feature vector are vector signals describing an image histogram.

29. A method of creating a summary of a plurality of video frames comprising the steps of:

(a) dividing said plurality of video frames into at least one video segment of relatively homogeneous content comprising at least one said video frame;
(b) defining feature vectors representative of each of said video frames;
(c) ranking said feature vectors representing said video frames included in said video segment according to a distortion produced in a set of said feature vectors representing said segment by each of said feature vectors in said set of feature vectors; and
(d) including in said summary said video frames represented by said feature vectors ranked as producing a minimum of said distortion.

30. The method of claim 29 wherein said relative distortion is determined with a vector rank filter (70).

31. The method of claim 29 further comprising the step of including in said summary said video frames represented by said feature vectors ranked as producing relative distortion greater than said minimum of said relative distortion.

32. The method of claim 29 wherein the step of dividing said plurality of video frames into at least one video segment comprises the steps of:

(a) defining a threshold distortion;
(b) locating a first frame in said video segment;
(c) defining a first feature vector representative of said first frame;
(d) including said first feature vector in a set of segment feature vectors;
(e) defining a next feature vector representative of a subsequent video frame;
(f) including said next feature vector with said set of segment feature vectors;
(g) calculating a distortion of said set of segment feature vectors resulting from including said next feature vector with said set of segment feature vectors;
(h) comparing said distortion of said set of segment feature vectors with said threshold distortion; and
(i) repeating steps (e) through (h) until said distortion of said set of segment feature vectors has a predefined relationship to said threshold distortion thereby defining a second boundary of said segment.

33. The method of claim 32 wherein said distortion of said set of segment feature vectors is determined with a vector rank filter (70).

34. The method of claim 32 wherein said threshold distortion has a predefined relationship to a minimum of said distortion.

35. A method of creating a summary of a plurality of video frames comprising the steps of:

(a) dividing said plurality of video frames into at least one video segment of relatively homogeneous content comprising at least one said video frame;
(b) defining a feature vector representative of each of said video frames;
(c) ranking said feature vectors according to a relative cumulative distance from each of said feature vectors to all other said feature vectors in a set of said feature vectors representing said video frames of said segment; and
(d) including in said summary said video frames represented by said feature vectors ranked as having a minimum of said relative cumulative distance.

36. The method of claim 35 wherein said relative cumulative distance is determined with a vector rank filter (70).

37. The method of claim 35 further comprising the step of including in said summary said video frames represented by said feature vectors ranked as being at a relative cumulative distance greater than said minimum of said relative cumulative distance.

38. The method of claim 35 wherein the step of dividing said plurality of video frames into at least one video segment comprises the steps of:

(a) defining a threshold value;
(b) locating a first frame in said video segment;
(c) defining a first feature vector representative of said first frame;
(d) including said first feature vector in a set of segment feature vectors;
(e) defining a next feature vector representative of a subsequent video frame;
(f) including said next feature vector with said set of segment feature vectors;
(g) calculating a cumulative distance from said next feature vector to each said feature vector in said set of segment feature vectors;
(h) updating said cumulative distance;
(i) comparing said updated cumulative distance with said threshold value; and
(j) repeating steps (e) through (i) until said updated cumulative distance has a predefined relationship to said threshold value thereby defining said second boundary of said video segment.

39. The method of claim 38 wherein a minimum of said cumulative distance is determined with a vector median filter (12).

40. The method of claim 38 wherein said threshold value has a predefined relationship to said minimum of said cumulative distance.

41. The method of claim 35 wherein said relative cumulative distance is determined with a vector clustering algorithm.

42. The method of claim 29 or 35 wherein said feature vectors are vector signals characterizing image histograms.

43. An apparatus (80) to skim a plurality of video frames (82) of relatively homogeneous content comprising:

(a) a receiver (84) to receive said plurality of frames (82);
(b) a vector converter (86) to characterize said plurality of frames (82) as a plurality of vector signals;
(c) a vector filter (90) to rank said vector signals according to a relative distortion produced by each of said vector signals in a set of all said vector signals; and
(d) a skimmer (94) to identify each of said frames corresponding to each of said vector signals producing said relative distortion of a specified rank.

44. An apparatus (80) to skim a plurality of video frames (82) of relatively homogeneous content comprising:

(a) a receiver (84) to receive said plurality of frames (82);
(b) a vector converter (86) to characterize said plurality of frames (82) as a plurality of vector signals;
(c) a vector filter (90) to rank said vector signals according to a relative cumulative distance from each of said vector signals to all other said vector signals in a set of all said vector signals; and
(d) a skimmer (94) to identify each of said frames (82) corresponding to each of said vector signals producing said relative cumulative distance of a specified rank.

45. The apparatus (80) of claim 43 or 44 wherein said vector signals characterize image histograms of said frames.

46. An apparatus (80) for identifying a last frame in a segment of video frames of relatively homogeneous content within a plurality of video frames (82) comprising:

(a) a receiver (84) to receive said plurality of frames (82);
(b) a vector converter (86) to characterize each of said plurality of frames (82) as a vector signal;
(c) an accumulator (88) to accumulate a set of said vector signals comprising said vector signal corresponding to a first frame in said segment and successively thereafter said vector signal corresponding to at least one subsequent frame;
(d) a vector filter (90) to calculate a distortion of said set of vector signals as each said vector signal is accumulated with said set of vector signals; and
(e) a comparator (92) to identify said frame (82) corresponding to said vector signal producing said distortion of said set of vector signals having a predefined relationship to a specified threshold distortion thereby identifying said last frame in said segment.

47. The method of claim 46 wherein said threshold distortion has a predefined relationship to a minimum of said distortion of said set of vector signals.

48. An apparatus (80) for identifying a last frame in a segment of video frames of relatively homogeneous content within a plurality of video frames (82) comprising:

(a) a receiver (84) to receive said plurality of frames (82);
(b) a vector converter (86) to characterize each of said plurality of frames (82) as a vector signal;
(c) an accumulator (88) to accumulate a set of said vector signals comprising said vector signal corresponding to a first frame in said segment and successively thereafter said vector signal corresponding to at least one subsequent frame;
(d) a vector filter (90) to calculate a relative cumulative distance from said vector signal to each other said vector signal of said set of vector signals as each said vector signal is accumulated with said set of vector signals; and
(e) a comparator (92) to identify said frame corresponding to said vector signal characterized by said relative cumulative distance having a predefined relationship to a specified threshold value thereby identifying said last frame in said segment.

49. The method of claim 48 wherein said threshold value has a predefined relationship to a minimum of said cumulative distance.

50. An apparatus (80) for producing a summary of a plurality of video frames (82) comprising:

(a) a receiver (84) to receive said plurality of frames (82);
(b) a vector converter (86) to characterize each of said plurality of frames (82) as a vector signal,
(c) an accumulator (88) to accumulate a set of said vector signals comprising said vector signal corresponding to a first frame in a segment of frames having relatively homogeneous content and successively thereafter said vector signal corresponding to at least one subsequent frame;
(d) a vector filter (90) to calculate a distortion in said set of vector signals as each said vector signal is accumulated in said set of vector signals and rank each said vector signal relative to each other said vector signal in said set of vector signals according to a relative measure of said distortion;
(e) a comparator (92) to identify said vector signal producing said distortion in said set of vector signals having a predefined relationship to a specified threshold distortion thereby identifying a last frame in said segment of frames (82) of relatively homogeneous content; and
(f) a skimmer (94) to identify each said frame (82) in said segment corresponding to each of said vector signal producing said distortion of a specified relative rank.

51. The apparatus (80) of claim 50 wherein said threshold distortion has a predefined relationship to a minimum distortion of said set of vector signals.

52. An apparatus (80) for producing a summary of a plurality of video frames (82) comprising:

(a) a receiver (84) to receive said plurality of frames (82);
(b) a vector converter (86) to characterize each of said plurality of frames (82) as a vector signal:
(c) an accumulator (88) to accumulate a set of said vector signals comprising said vector signal corresponding to a first frame in a segment of frames having relatively homogeneous content and successively thereafter said vector signal corresponding to at least one subsequent frame;
(d) a vector filter (90) to calculate a cumulative distance from each said vector signal to each other said vector signal of said set of vector signals as each said vector signal is accumulated in said set of vector signals; and rank each said vector signal relative to each other said vector signal in said set of vector signals according to a relative measure of said cumulative distance;
(e) a comparator (92) to identify said vector signal characterized by said relative cumulative distance having a predefined relationship to a specified threshold value thereby identifying a last frame in said segment of frames of relatively homogeneous content; and
(f) a skimmer (94) to identify each said frame in said segment corresponding to each said vector signal characterized by said relative cumulative distance of a specified relative rank.

53. The apparatus (80) of claim 52 wherein said threshold distortion has a predefined relationship to a minimum cumulative distance for said set of vector signals.

54. The apparatus (80) of claim 50 or 52 wherein said vector signal characterizes an image histogram of said video frame.

55. A method of creating a summary of a plurality of video frames comprising the steps of:

(a) receiving a first frame;
(b) receiving a subsequent frame;
(c) resolving a homogeneity of content of said first frame and each said subsequent frame received;
(d) comparing said homogeneity of said first frame and each said subsequent frame with a threshold homogeneity;
(e) repeating steps (b) through (d) until said homogeneity has some predefined relationship with said threshold homogeneity; and
(f) identifying at least one key frame from the set of said first frame and each said subsequent frame received having content that represents a minimum distortion within said set of said first frame and each said subsequent frame.

56. The method of claim 55 wherein said homogeneity of content of said frames is resolved with a vector rank filter (70).

57. The method of claim 55 wherein said homogeneity of content is characterized by a distortion of a vector signal.

58. A method of creating a summary of a plurality of video frames comprising the steps of:

(a) receiving a first frame;
(b) receiving a subsequent frame;
(c) resolving a homogeneity of content of said first frame and each said subsequent frame received;
(d) comparing said homogeneity of said first frame and each said subsequent frame with a threshold homogeneity;
(e) repeating steps (b) through (d) until said homogeneity has some predefined relationship with said threshold homogeneity; and
(f) identifying at least one key frame from the set of said first frame and each said subsequent frame received having content that represents a minimum cumulative distance to said set of said first frame and each said subsequent frame.

59. The method of claim 58 wherein said homogeneity of content of said frames is resolved with a vector median filter (12).

60. The method of claim 58 wherein said homogeneity of content is characterized by a cumulative distance from a vector signal to all other vector signals in said set of vector signals.

FIG.1

FIG.2A

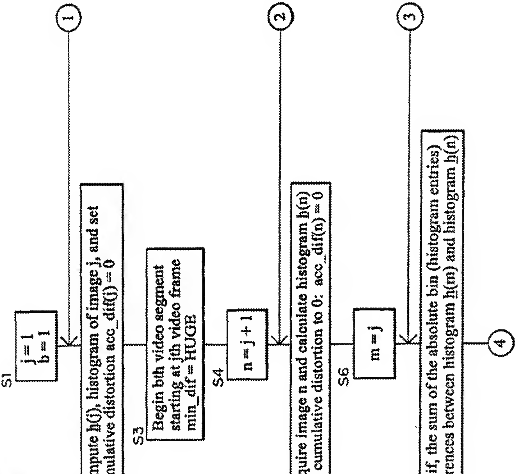

S1
j = 1
b = 1
1
S2
Compute h(j), histogram of image j, and set cumulative distortion acc_dif(j) = 0
S3
Begin bth video segment starting at jth video frame min_dif = HUGE
S4
n = j + 1
2
S5
Acquire image n and calculate histogram h(n)
Set cumulative distortion to 0: acc_dif(n) = 0
S6
m = j
3
S7
Compute dif, the sum of the absolute bin (histogram entries) value differences between histogram h(m) and histogram h(n)
4

FIG.2B

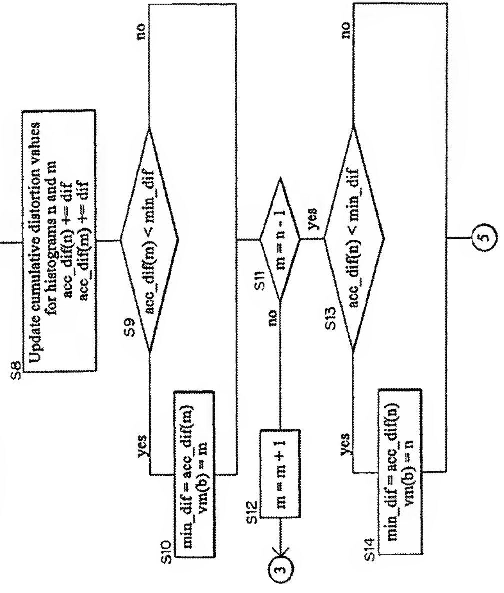

4
S8
Update cumulative distortion values
for histograms n and m
acc_dif(n) += dif
acc_dif(m) += dif
S9
acc_dif(m) < min_dif
yes
no
S10
min_dif = acc_dif(m)
vm(b) = m
S11
m = n - 1
no
yes
S12
m = m + 1
3
S13
acc_dif(n) < min_dif
yes
no
S14
min_dif = acc_dif(n)
vm(b) = n
5

FIG.2C

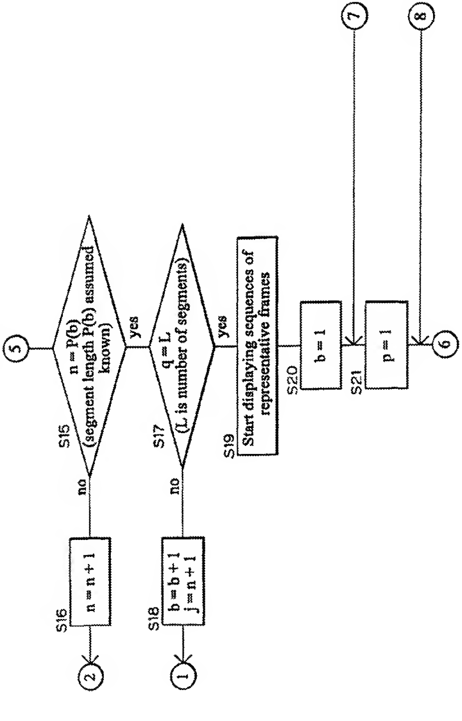

5
S15
n = P(b)
(segment length P(b) assumed known)
no
yes
S16
n = n + 1
2
S17
q = L
(L is number of segments)
no
yes
S18
b = b + 1
j = n + 1
1
S19
Start displaying sequences of representative frames
S20
b = 1
S21
p = 1
7
8
6

FIG.2D

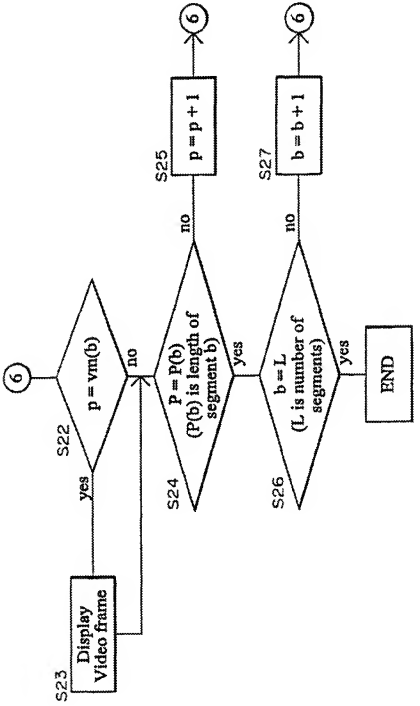

6
S22
p = vm(b)
yes
no
S23
Display
Video frame
S24
p = P(b)
(P(b) is length of
segment b)
yes
no
S25
p = p + 1
6
S26
b = L
(L is number of
segments)
yes
no
S27
b = b + 1
6
END

FIG.3

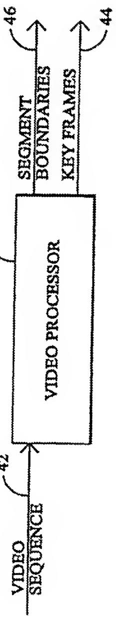

VIDEO SEQUENCE
42
VIDEO PROCESSOR
40
SEGMENT BOUNDARIES
46
KEY FRAMES
44

FIG.4A

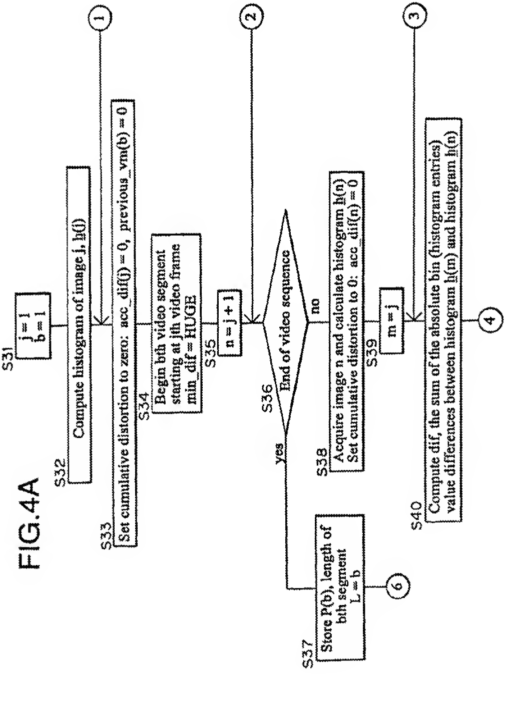

S31
j = 1
b = 1
S32
Compute histogram of image j, h(j)
1
S33
Set cumulative distortion to zero: acc_dif(j) = 0, previous_vm(b) = 0
S34
Begin bth video segment
starting at jth video frame
min_dif = HUGE
S35
n = j + 1
2
S36
End of video sequence
yes
no
S37
Store P(b), length of
bth segment
L = b
6
S38
Acquire image n and calculate histogram h(n)
Set cumulative distortion to 0: acc_dif(n) = 0
S39
m = j
3
S40
Compute dif, the sum of the absolute bin (histogram entries)
value differences between histogram h(m) and histogram h(n)
4

FIG.4B

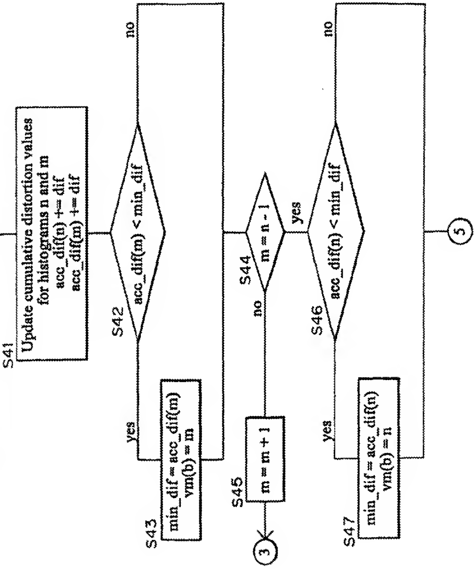

4
S41
Update cumulative distortion values
for histograms n and m
acc_dif(n) += dif
acc_dif(m) += dif
S42
acc_dif(m) < min_dif
yes
no
S43
min_dif = acc_dif(m)
vm(b) = m
S44
m = n - 1
no
yes
S45
m = m + 1
3
S46
acc_dif(n) < min_dif
yes
no
S47
min_dif = acc_dif(n)
vm(b) = n
5

FIG.4C

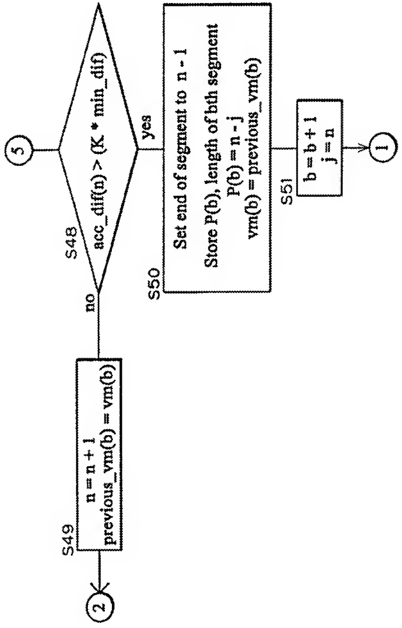

5
S48
acc_dif(n) > (K * min_dif)
yes
no
S50
Set end of segment to n - 1
Store P(b), length of bth segment
P(b) = n - j
vm(b) = previous_vm(b)
S51
b = b + 1
j = n
1
S49
n = n + 1
previous_vm(b) = vm(b)
2

FIG.4D

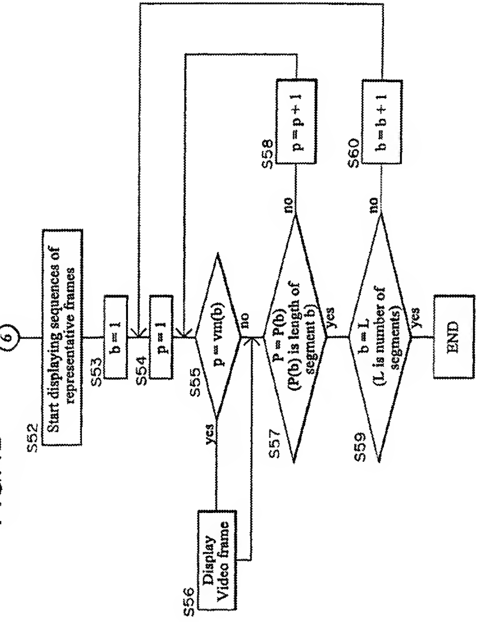

6
S52
Start displaying sequences of representative frames
S53
b = 1
S54
p = 1
S55
p = vm(b)
yes
no
S56
Display Video frame
S57
P = P(b)
(P(b) is length of segment b)
no
yes
S58
p = p + 1
S59
b = L
(L is number of segments)
no
yes
S60
b = b + 1
END

FIG.5

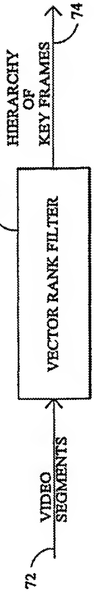

VIDEO SEGMENTS
72
VECTOR RANK FILTER
70
HIERARCHY OF KEY FRAMES
74

FIG.6

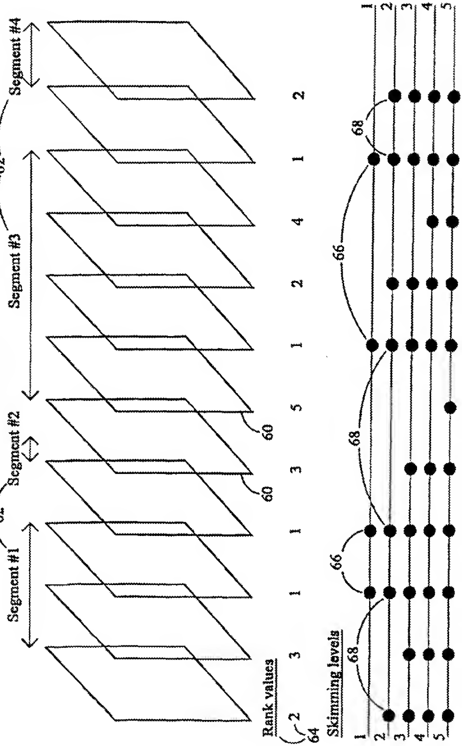

Segment #1
Segment #2
Segment #3
Segment #4
62
60
64
66
68
Rank values
2 3 1 1 3 5 1 2 4 1 2
Skimming levels
1
2
3
4
5

FIG.7A

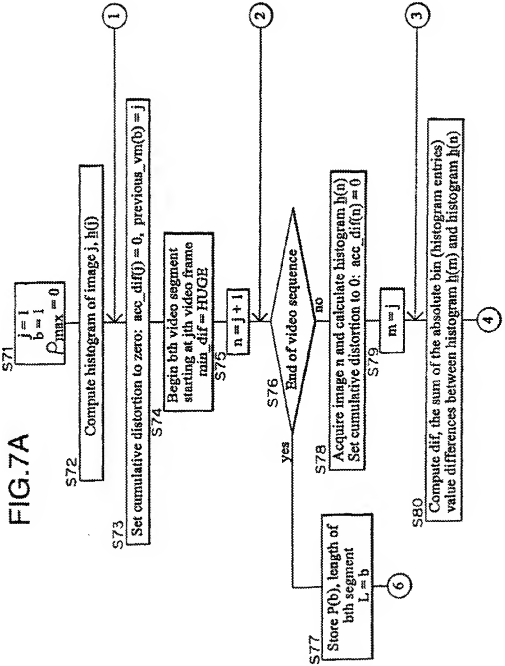

S71
j = 1
b = 1
$\rho_{max} = 0$
S72
Compute histogram of image j, h(j)
S73
Set cumulative distortion to zero: acc_dif(j) = 0, previous_vm(b) = j
S74
Begin bth video segment
starting at jth video frame
min_dif = HUGE
S75
n = j + 1
S76
End of video sequence
yes
no
S77
Store P(b), length of
bth segment
L = b
S78
Acquire image n and calculate histogram h(n)
Set cumulative distortion to 0: acc_dif(n) = 0
S79
m = j
S80
Compute dif, the sum of the absolute bin (histogram entries)
value differences between histogram h(m) and histogram h(n)
1
2
3
4
6

FIG.7B

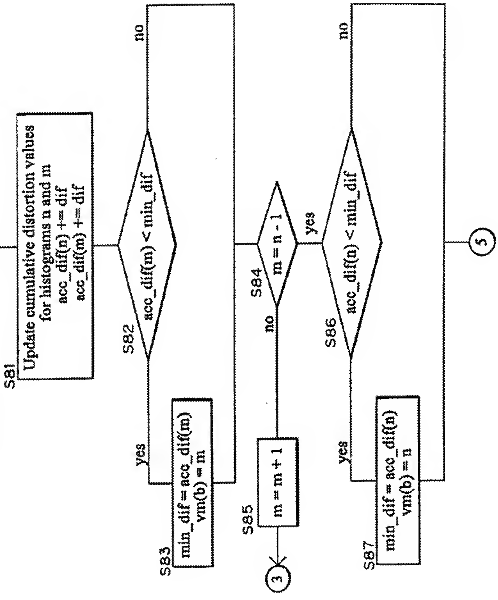

4
S81
Update cumulative distortion values
for histograms n and m
acc_dif(n) += dif
acc_dif(m) += dif
S82
acc_dif(m) < min_dif
yes
no
S83
min_dif = acc_dif(m)
vm(b) = m
S84
m = n - 1
yes
no
S85
m = m + 1
3
S86
acc_dif(n) < min_dif
yes
no
S87
min_dif = acc_dif(n)
vm(b) = n
5

FIG.7C

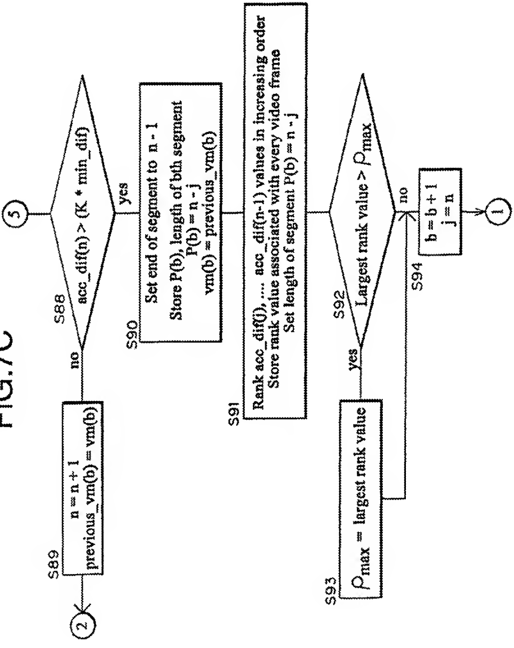

5
S88
acc_dif(n) > (K * min_dif)
yes
no
S89
n = n + 1
previous_vm(b) = vm(b)
2
S90
Set end of segment to n - 1
Store P(b), length of bth segment
P(b) = n - j
vm(b) = previous_vm(b)
S91
Rank acc_dif(j), ..., acc_dif(n-1) values in increasing order
Store rank value associated with every video frame
Set length of segment P(b) = n - j
S92
Largest rank value > ρ_{max}
yes
no
S93
ρ_{max} = largest rank value
S94
b = b + 1
j = n
1

FIG.7D

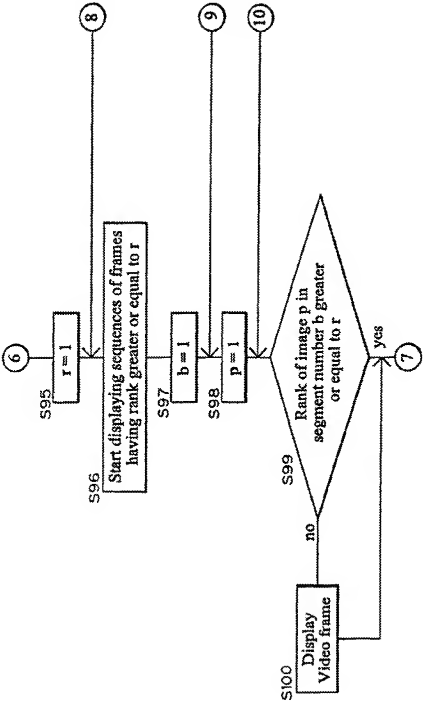

6
S95
r = 1
8
S96
Start displaying sequences of frames having rank greater or equal to r
S97
b = 1
9
S98
p = 1
10
S99
Rank of image p in segment number b greater or equal to r
no
yes
7
S100
Display Video frame

FIG.7E

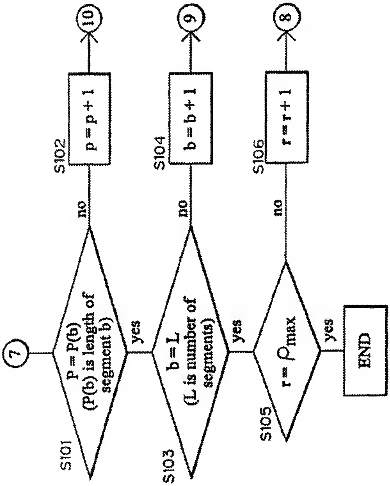
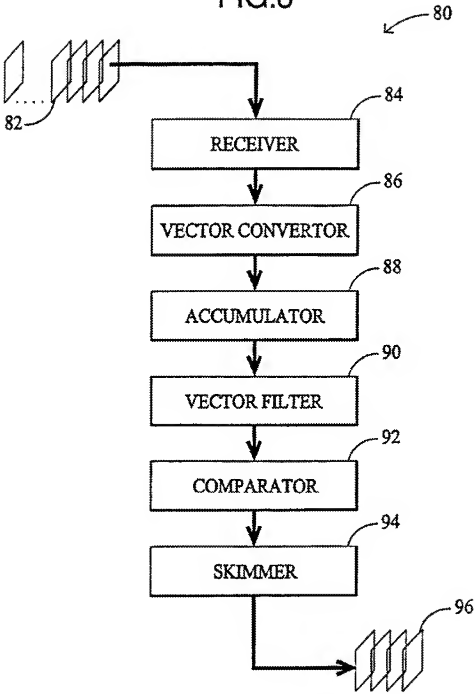

7
S101
P = P(b)
(P(b) is length of segment b)
no
S102
p = p + 1
10
yes
S103
b = L
(L is number of segments)
no
S104
b = b + 1
9
yes
S105
r = ρmax
no
S106
r = r + 1
8
yes
END

FIG.8


80
82
84
RECEIVER
86
VECTOR CONVERTOR
88
ACCUMULATOR
90
VECTOR FILTER
92
COMPARATOR
94
SKIMMER
96